GWT forschung+innovation

Vampir 7

User Manual

Copyright

Support / Feedback / Bugreports

Please provide us feedback! We are very interested to hear what people like, dislike, or what features they are interested in.

If you experience problems or have suggestions about this application or manual, please contact service@vampir.eu.

When reporting a bug, please include as much detail as possible, in order to reproduce it. Please send the version number of your copy of *Vampir* along with the bug report. The version is stated in the “About Vampir” dialog accessible from the main menu under “Help → About Vampir”.

Please visit http://vampir.eu for updates.

service@vampir.eu
http://vampir.eu

Manual Version

2011-11-11 / Vampir 7.5

Contents

1 Introduction

Performance optimization is a key issue for the development of efficient parallel software applications. *Vampir* provides a manageable framework for analysis, which enables developers to quickly display program behavior at any level of detail. Detailed performance data obtained from a parallel program execution can be analyzed with a collection of different performance views. Intuitive navigation and zooming are the key features of the tool, which help to quickly identify inefficient or faulty parts of a program code. *Vampir* implements optimized event analysis algorithms and customizable displays which enable a fast and interactive rendering of very complex performance monitoring data. Ultra large data volumes can be analyzed with a parallel version of *Vampir*, which is available on request.

Vampir has a product history of more than 15 years and is well established on Unix based HPC systems. This tool experience is also available for HPC systems that are based on *Microsoft Windows HPC Server 2008*.

1.1 Event-based Performance Tracing and Profiling

In software analysis, the term profiling refers to the creation of tables, which summarize the runtime behavior of programs by means of accumulated performance measurements. Its simplest variant lists all program functions in combination with the number of invocations and the time that was consumed. This type of profiling is also called inclusive profiling, as the time spent in subroutines is included in the statistics computation.

A commonly applied method for analyzing details of parallel program runs is to record so-called trace log files during runtime. The data collection process itself is also referred to as tracing a program. Unlike profiling, the tracing approach records timed application events like function calls and message communication as a combination of timestamp, event type, and event specific data. This creates a stream of events, which allows very detailed observations of parallel programs. With this technology, synchronization and communication patterns of parallel program runs can be traced and analyzed in terms of performance and correctness. The analysis is usually carried out in a postmortem step, i.e., after completion of the program. It is needless to say that program traces can also be used to calculate the profiles mentioned above. Computing profiles from trace data allows arbitrary time intervals and process groups to be specified. This is in contrast to “fixed” profiles accumulated during runtime.

1.2 The Open Trace Format (OTF)

The *Open Trace Format* (OTF) was designed as a well-defined trace format with open, public domain libraries for writing and reading. This open specification of the trace information provides analysis and visualization tools like *Vampir* to operate efficiently at large scale. The format addresses large applications written in an arbitrary combination of Fortran77, Fortran (90/95/etc.), C, and C++.

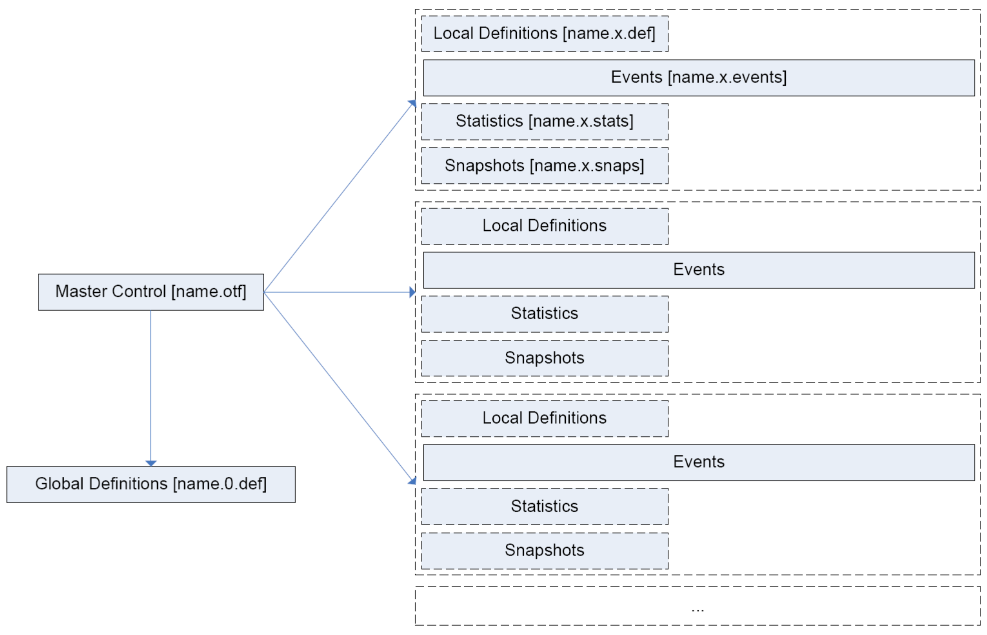


Figure 1.1: Representation of Streams by Multiple Files

OTF uses a special ASCII data representation to encode its data items with numbers and tokens in hexadecimal code without special prefixes. That enables a very powerful format with respect to storage size, human readability, and search capabilities on timed event records.

In order to support fast and selective access to large amounts of performance trace data, OTF is based on a stream-model, i.e. single separate units representing segments of the overall data. OTF streams may contain multiple independent processes whereas a process belongs to a single stream exclusively. As shown in Figure 1.1, each stream is represented by multiple files which store definition records, performance events, status information, and event summaries separately. A single global master file holds the necessary information for the process to stream mappings.

Each file name starts with an arbitrary common prefix defined by the user. The master file is always named {*name*}*.otf*. The global definition file is named {*name*}*.0.def*. Events and local definitions are placed in files {*name*}*.x.events* and {*name*}*.x.defs* where the latter files are optional. Snapshots and statistics are placed in files named {*name*}*.x.snaps* and {*name*}*.x.stats* which are optional, too.

Note: Open the master file (**.otf*) to load a trace. When copying, moving or deleting traces it is important to take all according files into account otherwise *Vampir* will render the whole trace invalid! Good practice is to hold all files belonging to one trace in a dedicated directory.

Detailed information about the *Open Trace Format* can be found in the ``Open Trace Format (OTF)''[1] documentation.

1.3 Vampir and Windows HPC Server 2008

The *Vampir* performance visualization tool usually consists of a performance monitor (*VampirTrace*) that records performance data and a performance GUI, which is responsible for the graphical representation of the data. In *Windows HPC Server 2008*, the performance monitor is fully integrated into the operating system, which simplifies its employment and provides access to a wide range of system metrics. A simple execution flag controls the generation of performance data. This is very convenient and an important difference to solutions based on explicit source, object, or binary modifications. *Windows HPC Server 2008* is shipped with a translator, which produces trace log files in *Vampir*'s *Open Trace Format* (OTF). The resulting files can be visualized with the *Vampir 7* performance data browser.

[1] http://www.tu-dresden.de/zih/otf

2 Getting Started

2.1 Installation of Vampir

Vampir is available on all major platforms but naturally its installation depends on the operation system.

2.1.1 Unix, Linux

In order to install *Vampir* on an Unix/Linux machine it is sufficient to unpack the tarball into the installation folder. After that, start the *Vampir* application and follow the instructions for license installation.

2.1.2 Mac OS X

Open the .dmg installation package and drag the *Vampir* icon into the applications folder on your computer. You might need administrator rights to do so. Alternatively, you can also drag the *Vampir* application to another directory that is writable for you. After that, double click on the *Vampir* application and follow the instructions for license installation.

2.1.3 Windows

On Windows platforms the provided *Vampir* installer makes the installation very simple and straightforward. Just run the installer and follow the installation wizard. Install *Vampir* in a folder of your choice, e.g.:

```
C:\Program Files
```

In order to run the installer in silent (unattended) mode use the */S* option. It is also possible to specify the output folder of the installation with */D=dir*. An example of a silent installation command is as follows:

```
Vampir-7.5.0-Standard-setup-x86.exe /S /D=C:\Program Files
```

If you want to, you can associate *Vampir* with OTF trace files (**.otf*) during the installation process. The *Open Trace Format* (OTF) is described in Chapter 1.2. This allows you to load a trace file quickly by double-clicking it. Subsequently, *Vampir* can be launched by double-clicking its icon or by using the command line interface (see Chapter 2.4).

At the first start *Vampir* will display instructions for license installation.

2.2 Generation of Trace Data on Windows Systems

2.2.1 Enabling Performance Tracing

The generation of trace log files for the *Vampir* performance visualization tool requires a working monitoring system to be attached to your parallel program.

The *Event Tracing for Windows* (ETW) infrastructure of the *Windows* client and server OS's is such a monitor. The *Windows HPC Server 2008* version of MS-MPI has built-in support for this monitor. It enables application developers to quickly produce traces in production environments by simply adding an extra mpiexec flag (`-trace`). In order to trace an application the user account is required to be a member of the "Administrator" or "Performance Log Users" groups. No special builds or administrative privileges are necessary. The cluster administrator will only have to add the "Performance Log Users" group to the head node's "Users" group, if you want to use this group for tracing. Trace files will be generated during the execution of your application. The recorded trace log files include the following events: Any MS-MPI application call and low-level communication within sockets, shared memory, and NetworkDirect implementations. Each event includes a high-precision CPU clock timer for precise visualization and analysis.

2.2.2 Tracing an MPI Application

The steps necessary for monitoring the MPI performance of an MS-MPI application are depicted in Figure 2.1. First the application needs to be available throughout all compute nodes in the cluster and has to be started with tracing enabled. The *Event Tracing for Windows* (ETW) infrastructure writes eventlogs (.etl files) containing the respective MPI events of the application on each compute node. In order to achieve consistent event data across all compute nodes clock corrections need to be applied. This step is performed after the successful run of the application using the Microsoft tool *mpicsync*. Now the eventlog files can be converted into OTF files with help of the tool *etl2otf*. The last necessary step is to copy the generated OTF files from the compute nodes into one shared directory. Then this directory includes all files needed by the *Vampir* performance GUI. The application performance can be analyzed now.

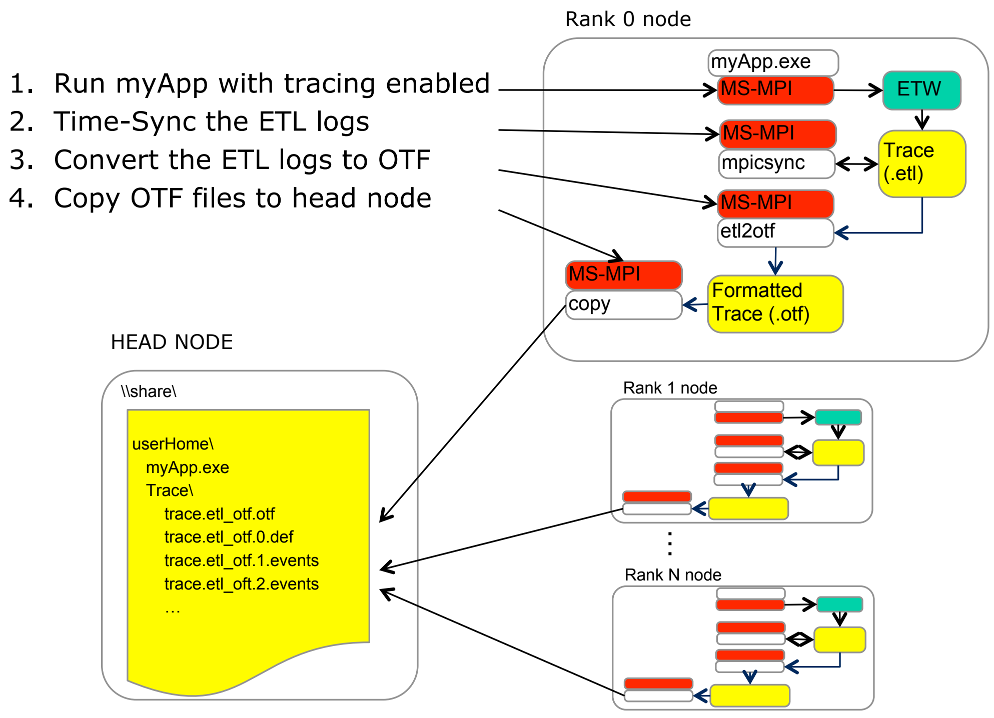


Figure 2.1: MS-MPI Tracing Overview

The following commands illustrate the procedure described above and show, as a practical example, how to trace an application on the Windows HPC Server 2008. For proper utilization and thus successful tracing, the file system of the cluster needs to meet the following prerequisites:

- `` `\\share\userHome' `` is the shared user directory throughout the cluster
- MS-MPI executable `myApp.exe` is available in the shared directory
- `` `\\share\userHome\Trace' `` is the directory where the OTF files are collected

1. Launch application with tracing enabled (use of `-tracefile` option):

```
mpiexec -wdir \\share\userHome\
-tracefile %USERPROFILE%\trace.etl myApp.exe
```

- `-wdir` sets the working directory; `myApp.exe` has to be there
- `%USERPROFILE%` translates to the local home directory, e.g. `` `C:\Users\userHome' ``; on each compute node the eventlog file (.etl) is stored locally in this directory

2. Time-sync the eventlog files throughout all compute nodes:

```
mpiexec -cores 1 -wdir %USERPROFILE% mpicsync trace.etl
```

- `-cores 1`: run only one instance of mpicsync on each compute node

3. Format the eventlog files to OTF files:

```
mpiexec -cores 1 -wdir %USERPROFILE% etl2otf trace.etl
```

4. Copy all OTF files from compute nodes to trace directory on share:

```
mpiexec -cores 1 -wdir %USERPROFILE% cmd /c copy /y
``*_otf*'' ``\\share\userHome\Trace''
```

2.3 Generation of Trace Data on Linux Systems

The generation of trace files for the (*Vampir*) performance visualization tool requires a working monitoring system to be attached to your parallel program.

Contrary to *Windows HPC Server 2008* - whereby the performance monitor is integrated into the operating system - recording performance under Linux is done by a separate performance monitor. We recommend our *VampirTrace* monitoring facility which is available as Open Source software.

During a program run of an application, *VampirTrace* generates an OTF trace file, which can be analyzed and visualized by *Vampir*. The *VampirTrace* library allows MPI communication events of a parallel program to be recorded in a trace file. Additionally, certain program-specific events can also be included. To record MPI communication events, simply relink the program with the *VampirTrace* library. A new compilation of the program source code is only necessary if program-specific events should be added.

Detailed information of the installation and usage of *VampirTrace* can be found in the `` `VampirTrace User Manual' ``[1].

2.3.1 Enabling Performance Tracing

To perform measurements with *VampirTrace*, the application program needs to be instrumented. *VampirTrace* handles this automatically by default, nevertheless manual instrumentation is possible as well.

All the necessary instrumentation of user functions, MPI, and OpenMP events is handled by the compiler wrappers of *VampirTrace* (vtcc, vtcxx, vtf77, vtf90 and the additional wrappers mpicc-vt, mpicxx-vt, mpif77-vt, and mpif90-vt in Open MPI 1.3).

All compile and link commands in the used makefile should be replaced by the *VampirTrace* compiler wrapper, which performs the necessary instrumentation of the program and links the suitable *VampirTrace* library.

[1]http://www.tu-dresden.de/zih/vampirtrace

Automatic instrumentation is the most convenient method to instrument your program. Therefore, simply use the compiler wrappers without any parameters, e.g.:

```
% vtf90 hello.f90 -o hello
```

For manual instrumentation with the *VampirTrace* API simply include ``vt_user.inc'' (Fortran) or ``vt_user.h'' (C, C++) and label any user defined sequence of statements for instrumentation as follows:

```
VT_USER_START(name) ...  VT_USER_END(name)
```

in Fortran and C, respectively in C++ as follows:

```
VT_TRACER(``name'');
```

Afterwards, use

```
% vtcc -DVTRACE hello.c -o hello
```

to combine the manual instrumentation with automatic compiler instrumentation or

```
% vtcc -vt:inst manual -DVTRACE hello.c -o hello
```

to prevent an additional compiler instrumentation.

For a detailed description of manual instrumentation, please consider the ``VampirTrace User Manual'' [2].

2.3.2 Tracing an Application

Running a *VampirTrace* instrumented application should normally result in an OTF trace file stored the current working directory where the application was executed. On Linux, Mac OS X, and Sun Solaris the default name of the trace file will be equal to the application name. For other systems, the default name is `a.otf` but can be defined manually by setting the environment variable `VT_FILE_PREFIX` to the desired name.

After a run of an instrumented application the traces of the single processes need to be unified in terms of timestamps and event IDs. In most cases, this happens automatically. If it is necessary to perform unification of local traces manually, use the following command:

```
% vtunify <nproc> <prefix>
```

If *VampirTrace* was built with support for OpenMP and/or MPI, it is possible to speedup the unification of local traces significantly. To distribute the unification on multiple processes the MPI parallel version vtunify-mpi can be used as follows:

```
% mpirun -np <nranks> vtunify-mpi <nproc> <prefix>
```

[2] http://www.tu-dresden.de/zih/vampirtrace

2.4 Starting Vampir and Loading a Trace File

Viewing performance data with the *Vampir* GUI is very easy. On Windows the tool can be started by double clicking its desktop icon (if installed) or by using the Start Menu. On a Linux-based machine run "./vampir" in the directory where *Vampir* is installed. A double click on the application icon opens *Vampir* on Mac OS X systems.

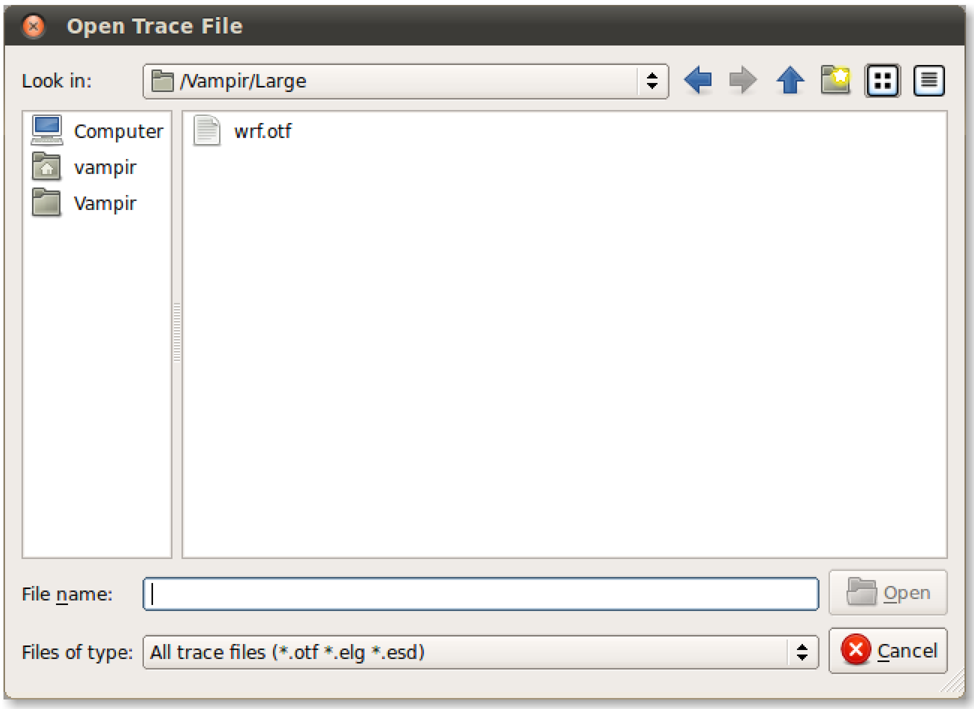


Figure 2.2: Loading a Trace Log File in Vampir

To open a trace file, select "Open. . . " in the "File" menu, which provides a file open dialog, depicted in Figure 2.2. It is possible to filter the files in the list. The file type input selector determines the visible files. The default "OTF Trace Files (*.otf)" shows only files that can be processed by the tool. All file types can be displayed by using "All Files (*)".

Alternatively, a command line invocation is always possible. Shown here is an example for a Windows system. Other platforms work accordingly.

```
C:\Program Files\Vampir\Vampir.exe [trace file]
```

To open multiple trace files at once you can give them one after another as command line arguments:

```
C:\Program Files\Vampir\Vampir.exe [file 1]...[file n]
```

If *Vampir* was associated with **.otf* files during the installation process, it is also possible to start the application by double-clicking an **.otf* file.

The trace files to be loaded have to be compliant with the *Open Trace Format* (OTF) standard (described in Chapter 1.2). *Microsoft HPC Server 2008* is shipped with the translator program *etl2otf.exe*, which produces appropriate input files for this platform.

While *Vampir* is loading the trace file, an empty "Trace View" window with a progress bar at the bottom opens. After *Vampir* loaded the trace data completely, a default set of charts will appear. The loading process can be interrupted at any time by clicking on the cancel button in the lower right corner of the "Trace View", depicted in Figure 2.3. Because events in the trace file are loaded one after another, the GUI will open but only show the earliest, already loaded information from the trace file. For large trace files with performance problems assumed to be at the beginning, this proceeding is a suitable strategy to save time.

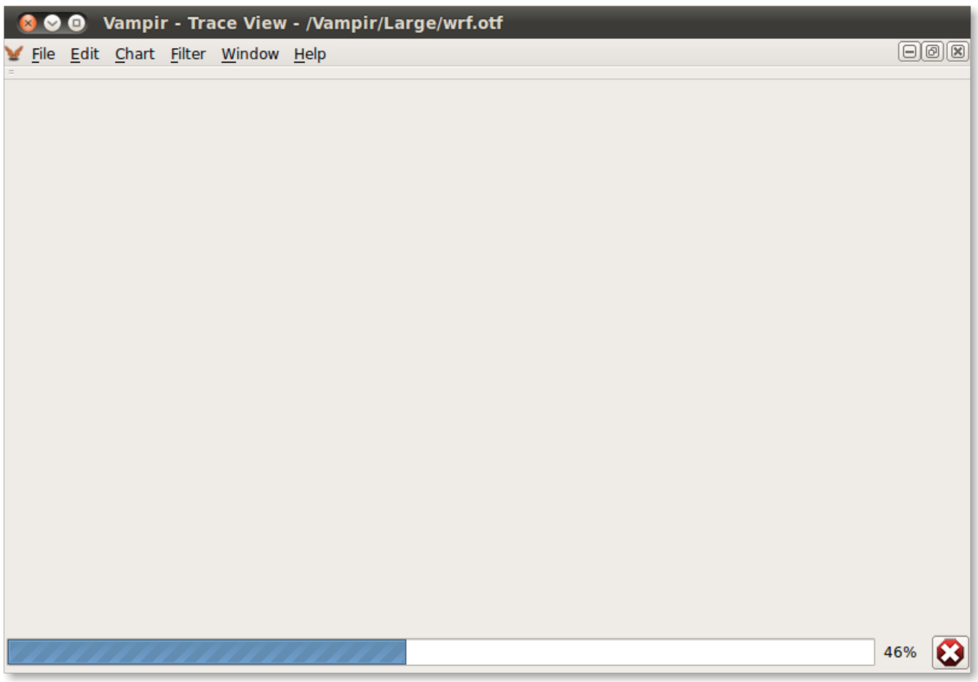


Figure 2.3: Progress Bar and Cancel Loading Button

The basic functionality and navigation elements of the GUI are described in Chapter 3. The available charts and the information provided by them are explained in Chapter 4.

3 Basics

After loading has been completed, the “Trace View” window title displays the trace file’s name as depicted in Figure 3.1. By default the “Charts” toolbar and the “Zoom Toolbar” are available.

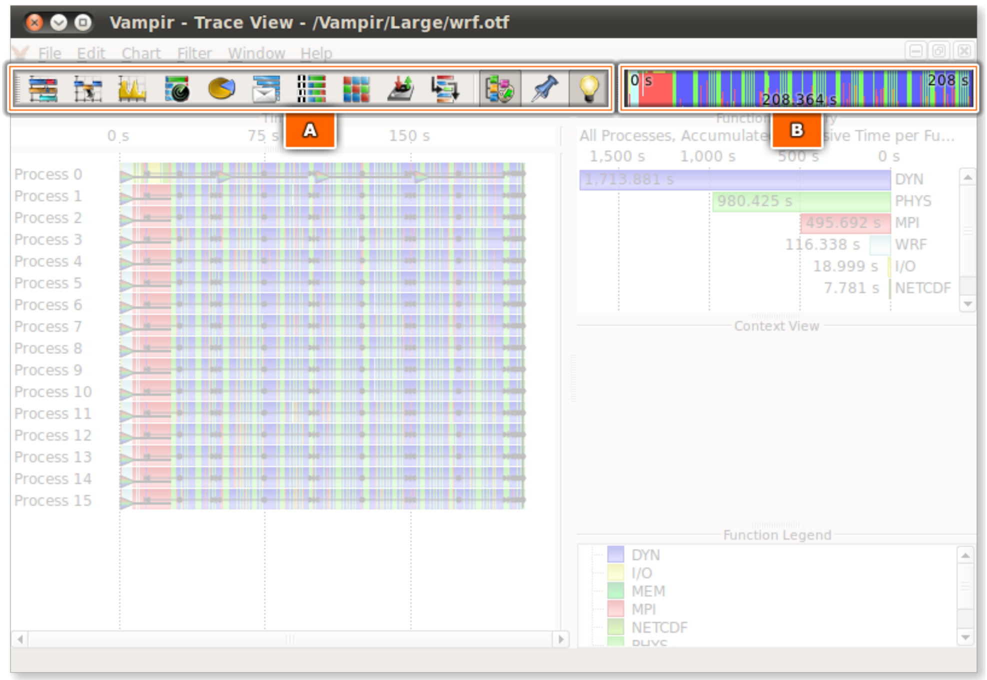

Vampir - Trace View - /Vampir/Large/wrf.otf
File Edit Chart Filter Window Help
A
B
0 s
208.364 s
208 s
All Processes, Accumulated Exclusive Time per Fu...
DYN
PHYS
MPI
WRF
I/O
NETCDF
Context View
Function Legend
MEM

Figure 3.1: Trace View Window with Charts Toolbar (A) and Zoom Toolbar (B)

Furthermore, a default set of charts is opened automatically after loading has been finished. The charts can be divided into three groups: timeline-, statistical-, and informational charts. Timeline charts show detailed event based information for arbitrary time intervals while statistical charts reveal accumulated measures which were computed from the corresponding event data. Informational charts provide additional or explanatory information regarding timeline- and statistical charts. All available charts can be opened with the “Charts” toolbar which is explained in Chapter 3.5.

In the following sections we will explain the basic functions of the *Vampir* GUI which are generic to all charts. If you are already familiar with the fundamentals feel free to skip this chapter. The details of the different charts are explained in Chapter 4.

3.1 Chart Arrangement

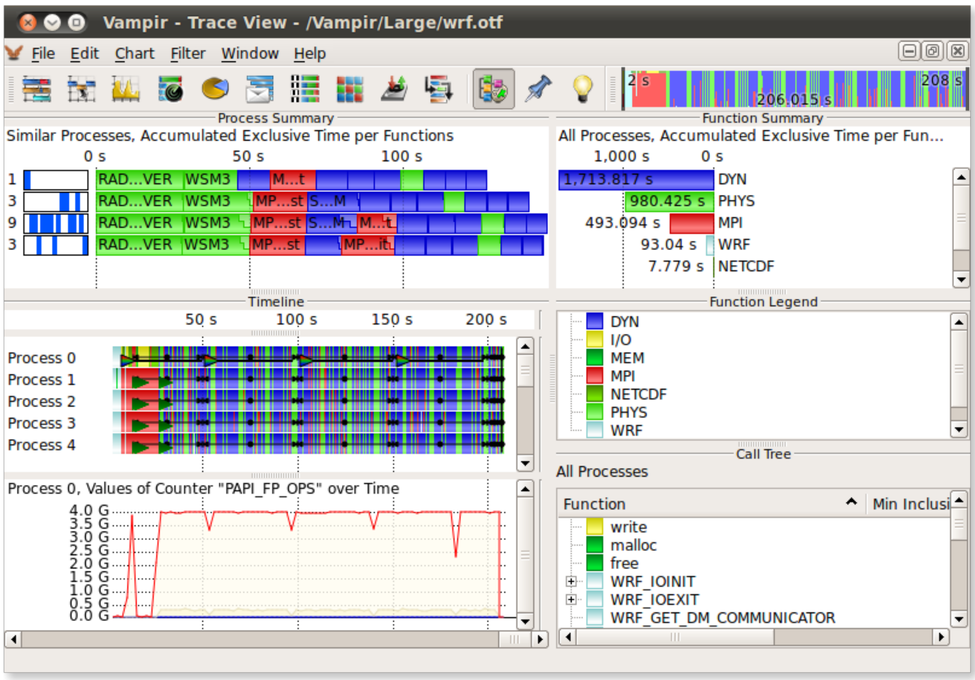


Figure 3.2: A Custom Chart Arrangement in the Trace View Window

The utility of charts can be increased by correlating them and their provided information. *Vampir* supports this mode of operation by allowing to display multiple charts at the same time. All timeline charts, such as the “Master Timeline” and the “Process Timeline” display a sequence of events. Those charts are therefore aligned vertically. This alignment ensures that the temporal relationship of events is preserved across chart boundaries.

The user can arrange the placement of the charts according to his preferences by dragging them into the desired position. When the left mouse button is pressed while the mouse pointer is located above a placement decoration, the layout engine will give visual clues as to where the chart may be moved. As soon as the user releases the left mouse button the chart arrangement will be changed according to his intentions. The entire procedure is depicted in Figures 3.3 and 3.4.

The layout engine furthermore allows a flexible adjustment of the screen space that is used by a chart. Charts of particular interest may get more space in order to render information in more detail.

The “Trace View” window can host an arbitrary number of charts. Charts can be added by clicking on the respective “Charts” toolbar icon or the corresponding “Chart” menu entry. With a few more clicks, charts can be combined to a custom chart arrangement

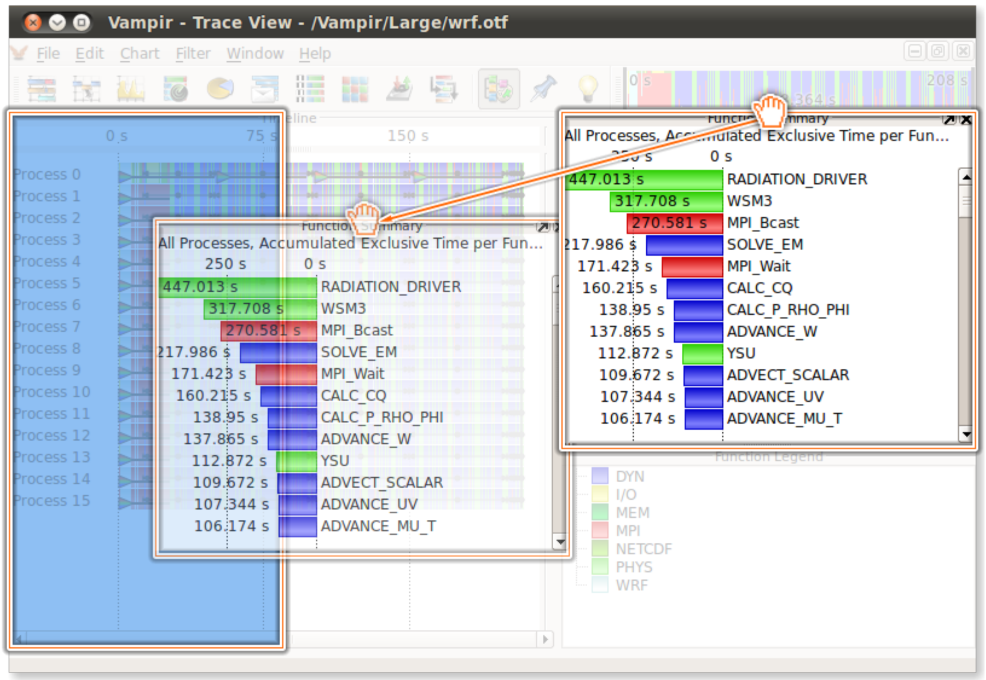


Figure 3.3: Moving and Arranging Charts in the Trace View Window (1)

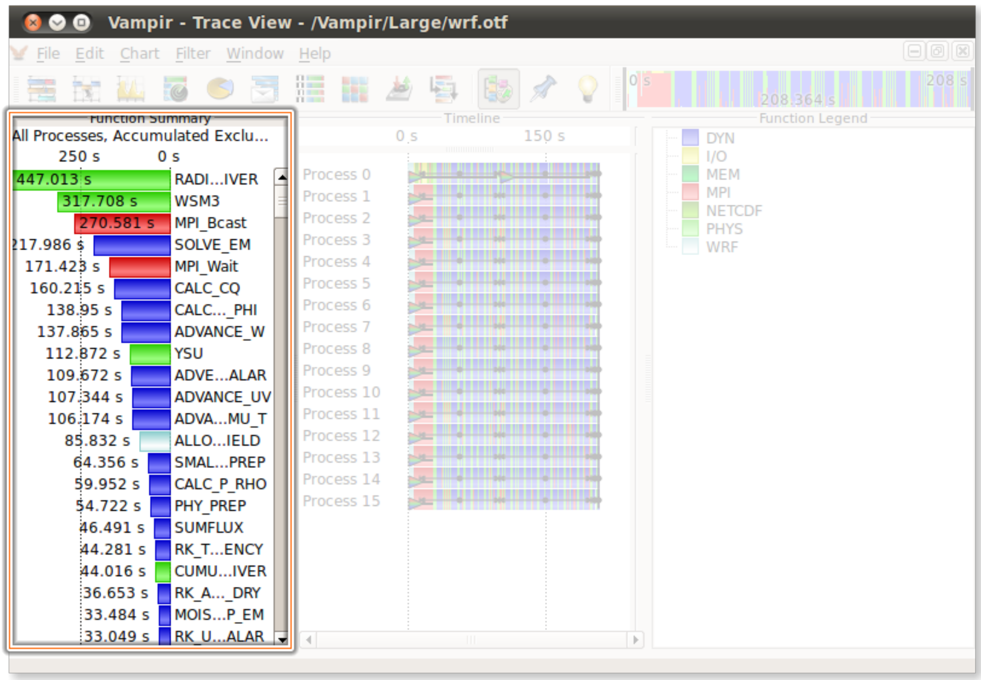


Figure 3.4: Moving and Arranging Charts in the Trace View Window (2)

as depicted in Figure 3.2. Customized layouts can be saved as described in Chapter 6.3.

Every chart can be undocked or closed by clicking the dedicated icon in its upper right corner as shown in Figure 3.5. Undocking a chart means to free the chart from the current arrangement and present it in an own window. To dock/undock a chart follow Figure 3.6, respectively Figure 3.7.

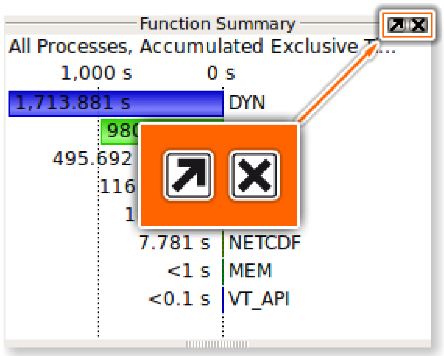


Figure 3.5: Closing (right) and Undocking (left) of a Chart

Considering that labels, e.g., those showing names or values of functions often need more space to show its whole text, there is a further option of resizing. In order to read labels completely, it might be useful to alter the distribution of space shared by the labels and the graphical representation in a chart. When hovering the blank space between labels and graphical representation, a movable separator appears. By dragging the separator decoration with the left mouse button the chart space provided for the labels can be resized. The whole process is illustrated in Figure 3.8.

3.2 Context Menus

All chart displays have their own context menu containing common as well as display specific entries. In this section only the most common entries will be discussed. A context menu can be accessed by right clicking anywhere in the chart window.

Common entries are:

- **Reset Zoom:** Go back to the initial state in horizontal zooming.
- **Reset Vertical Zoom:** Go back to the initial state in vertical zooming.
- **Set Metric:** Set the values which should be represented in the chart, e.g., change from “Exclusive Time” to “Inclusive Time”.
- **Sort By:** Rearrange values or bars by a certain characteristic.

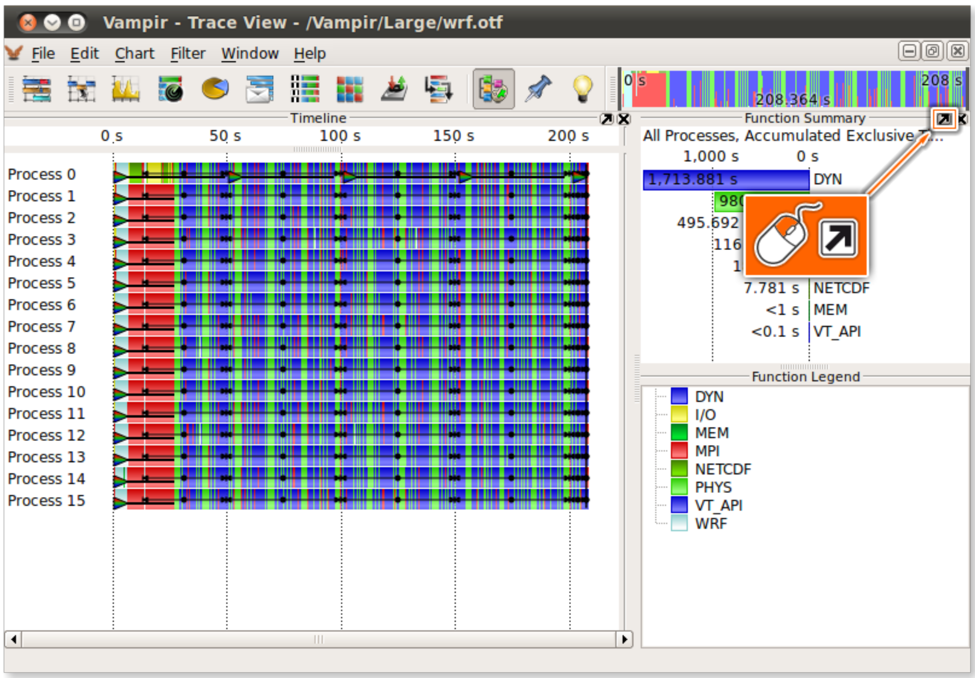

Vampir - Trace View - /Vampir/Large/wrf.otf
File Edit Chart Filter Window Help
Timeline
Function Summary
All Processes, Accumulated Exclusive T...
Function Legend
DYN
I/O
MEM
MPI
NETCDF
PHYS
VT_API
WRF

Figure 3.6: Undocking of a Chart

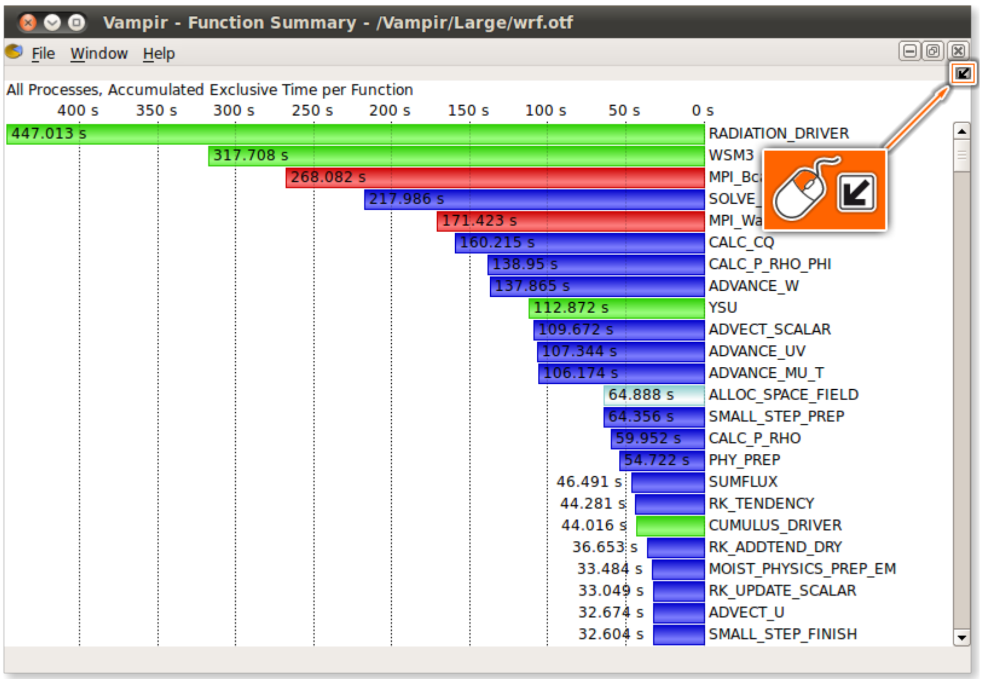


Figure 3.7: Docking of a Chart

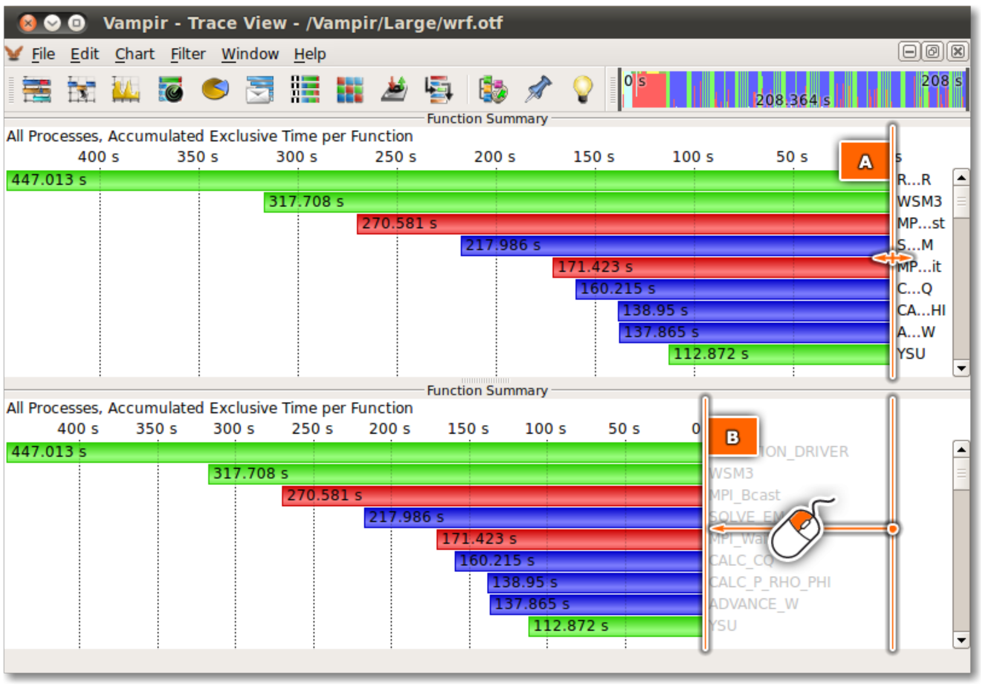


Figure 3.8: Resizing Labels: (A) Hover a Separator Decoration; (B) Drag and Drop the Separator

3.3 Zooming

Zooming is a key feature of *Vampir*. In most charts it is possible to zoom in and out to get detailed or abstract views of the visualized data.

In the timeline charts zooming produces a more detailed view of a selected time interval and therefore reveals new information that was previously hidden in the larger section. Short function calls in the “Master Timeline” may not be visible unless an appropriate zooming level has been reached. In other words, if the execution time of functions is too short with respect to the available pixel resolution of your computer display, zooming into a shorter time interval is required in order to make them visible.

Note: Other charts are affected by zooming in the timeline displays. The interval chosen in a timeline chart, such as “Master Timeline” or “Process Timeline” also defines the time interval for the calculation of accumulated measurements in the statistical charts.

Statistical charts like the “Function Summary” provide zooming of statistic values. In these cases zooming does not affect any other chart. Zooming is disabled in the “Pie Chart” mode of the “Function Summary” accessible via the context menu under “Set Chart Mode → Pie Chart”.

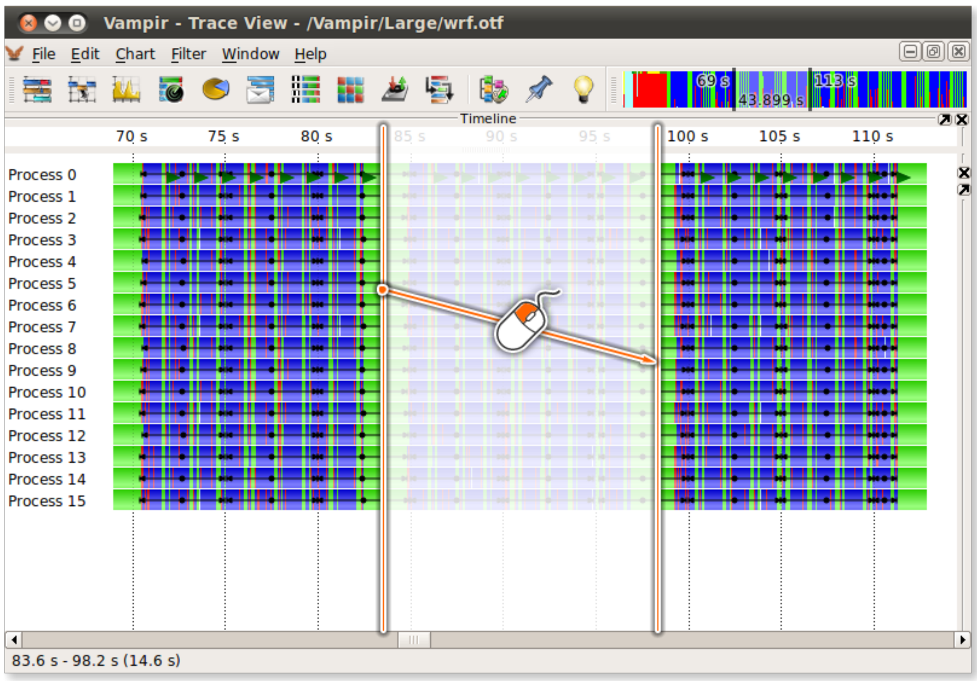


Figure 3.9: Zooming within a Chart

To zoom into an area, click and hold the left mouse button and select the area as shown in Figure 3.9. It is possible to zoom horizontally and in some charts also vertically. In the "Master Timeline" horizontal zooming defines the time interval to be visualized whereas vertical zooming selects a group of processes to be displayed. To scroll horizontally move the slider at the bottom or use the mouse wheel. To get back to the initial state of zooming select "Reset Horizontal Zoom" or "Reset Vertical Zoom" (see Section 3.2) in the context menu of the respective performance chart.

Additionally the zoom can be accessed with help of the "Zoom Toolbar" by dragging the borders of the selection rectangle or by scrolling of the mouse wheel as described in Chapter 3.4.

In order to return to the previous zooming state an *Undo* functionality, accessible via the "Edit" menu, is provided. Alternatively, the key combination *Ctrl+Z* also reverts the last zoom. Accordingly, a reverted zooming action can be redone by selecting *Redo* in the "Edit" menu or by pressing *Ctrl+Shift+Z*. The *Undo* functionality is not bound to single performance charts, but works across the entire *Vampir* application. The labels of the *Undo* and *Redo* menu entries also state which kind of action will be undone/redone next.

3.4 The Zoom Toolbar

Vampir provides a “Zoom Toolbar” that can be used for zooming and navigation in the trace data. It is located in the upper right corner of the “Trace View” window, shown in Figure 3.1. It is possible to adjust its position via drag and drop. The “Zoom Toolbar” offers an overview and summary of the loaded trace data. The currently zoomed area is highlighted as a rectangle within the “Zoom Toolbar”. By dragging of the two boundaries of the highlighted rectangle the horizontal zooming state can be adjusted.

Note: Instead of dragging boundaries it is also possible to use the mouse wheel for zooming. Hover the “Zoom Toolbar” and scroll up and down to zoom in and out, respectively.

Dragging the zoom area changes the section that is displayed without changing the zoom factor. For dragging, click into the highlighted zoom area and drag and drop it to the desired position. Zooming and dragging within the “Zoom Toolbar” is illustrated in Figure 3.10. If the user double clicks in the “Zoom Toolbar”, the initial zooming state is reverted.

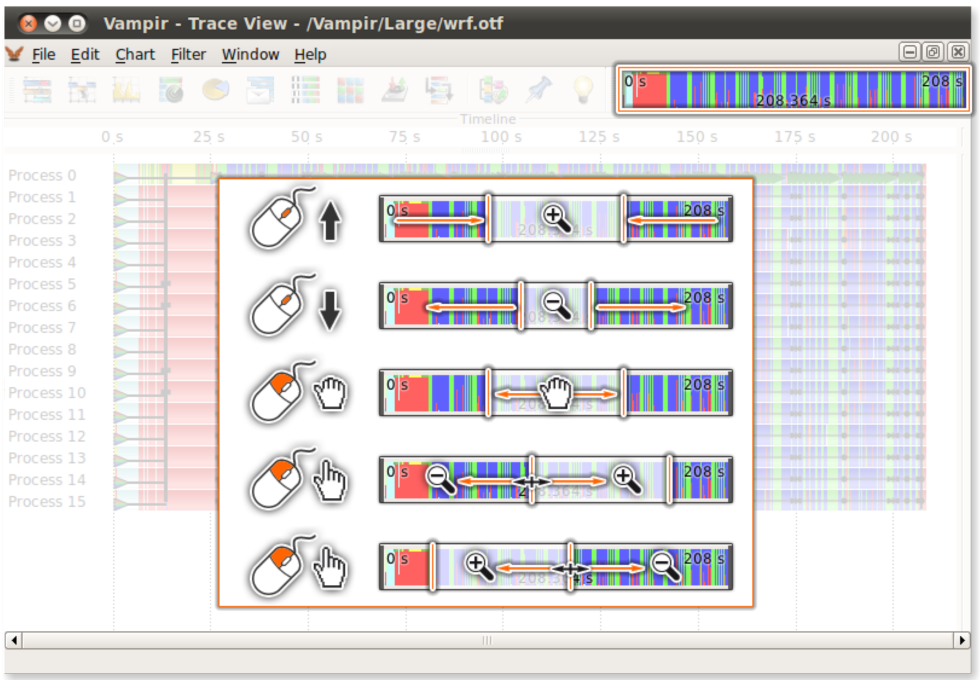


Figure 3.10: Zooming and Navigation within the Zoom Toolbar: (A+B) Zooming in/out with the Mouse Wheel; (C) Scrolling by Moving the Highlighted Zoom Area; (D) Zooming by Selecting and Moving a Boundary of the Highlighted Zoom Area

The colors represent user-defined groups of functions or activities. Please note that all charts added to the “Trace View” window will calculate their statistic information according to the selected time interval (zooming state) in the “Zoom Toolbar”. The “Zoom Toolbar” can be enabled and disabled with the toolbar’s context menu entry “Zoom Toolbar”.

3.5 The Charts Toolbar

Icon	Name	Description
▤	Master Timeline	Section 4.1.1
▤	Process Timeline	Section 4.1.1
📈	Counter Data Timeline	Section 4.1.2
📡	Performance Radar	Section 4.1.3
◔	Function Summary	Section 4.2.1
✉	Message Summary	Section 4.2.3
▦	Process Summary	Section 4.2.2
▦	Communication Matrix View	Section 4.2.4
💿	I/O Summary	Section 4.2.5
⇄	Call Tree	Section 4.2.6
🎨	Function Legend	Section 4.3.1
💡	Context View	Section 4.3.3
📌	Marker View	Section 4.3.2

Table 3.1: Icons of the Charts Toolbar

The “Charts Toolbar” is used to open instances of the available performance charts. It is located in the upper left corner of the “Trace View” window as shown in Figure 3.1. The toolbar can be dragged and dropped to alternative positions. The “Charts Toolbar” can be disabled with the toolbar’s context menu entry “Charts”.

Table 3.1 gives an overview of the available performance charts with their corresponding icons. The icons are arranged in three groups, divided by small separators. The first group represents timeline charts, whose zooming states affect all other charts. The second group consists of statistical charts, providing special information and statistics for a chosen interval. *Vampir* allows multiple instances for charts of these categories. The last group comprises of informational charts, providing specific textual information or legends. Only one instance of an informational chart can be opened at a time.

3.6 Properties of the Trace File

Vampir provides an info dialog containing important characteristics of the opened trace file. This “Trace Properties” dialog can be accessed via the main menu under “File $\rightarrow$ Get Info”. The information originates from the trace file and includes details such as file name, creator, or the OTF version.

4 Performance Data Visualization

This chapter deals with the different charts that can be used to analyze the behavior of a program and the comparison between different function groups, e.g. *MPI* and *Calculation*. Communication performance issues are regarded in this chapter as well. Various charts address the visualization of data transfers between processes. The following sections describe them in detail.

4.1 Timeline Charts

A very common chart type used in event-based performance analysis is the so-called timeline chart. This chart type graphically presents the chain of events of monitored processes or counters on a horizontal time axis. Multiple timeline chart instances can be added to the “Trace View” window via the “Chart” menu or the “Charts” toolbar.

Note: To measure the duration between two events in a timeline chart *Vampir* provides a tool called *Ruler*. In order to use the *Ruler* hold the “Shift” key pressed while clicking on any point of interest in a timeline chart and moving the mouse while holding the left mouse button pressed. A ruler like pattern appears in the timeline chart which provides the exact time between the start point and the current mouse position.

4.1.1 Master Timeline and Process Timeline

In the “Master Timeline” and the “Process Timeline” detailed information about functions, communication, and synchronization events is shown. Timeline charts are available for individual processes (“Process Timeline”) as well as for a collection of processes (“Master Timeline”). The “Master Timeline” consists of a collection of rows. Each row represents a single process, as shown in Figure 4.1. A “Process Timeline” shows the different levels of function calls in a stacked bar chart for a single process as depicted in Figure 4.2.

Every timeline row consists of a process name on the left and a colored sequence of function calls or program phases on the right. The color of a function is defined by its group membership, e.g., *MPI_Send()* belonging to the function group *MPI* has the same color, presumably red, as *MPI_Recv()*, which also belongs to the function group *MPI*. Clicking on a function highlights it and causes the “Context View” display to

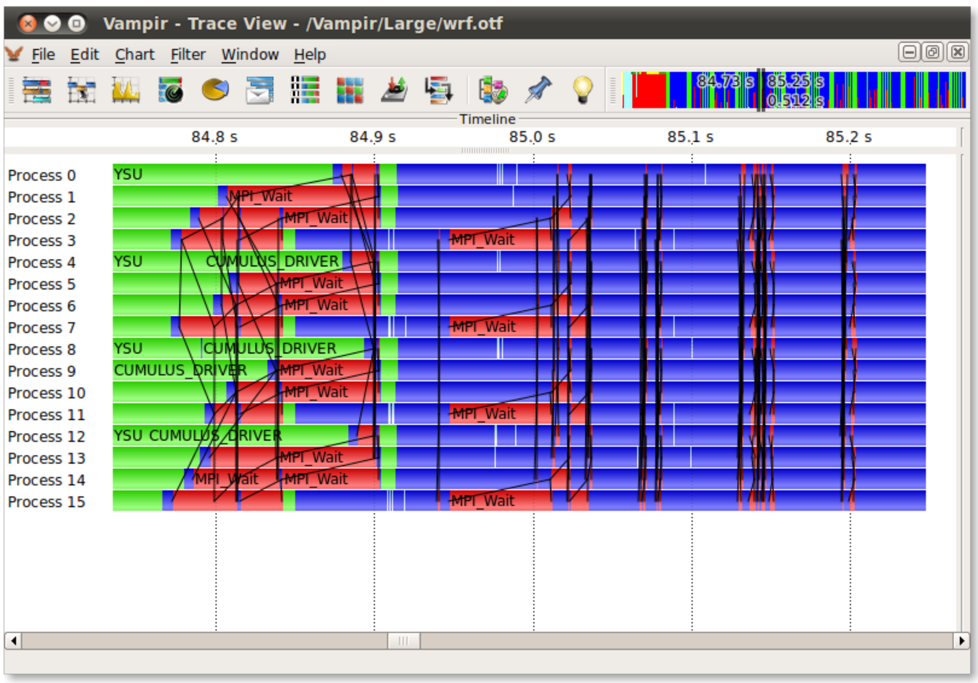


Figure 4.1: Master Timeline

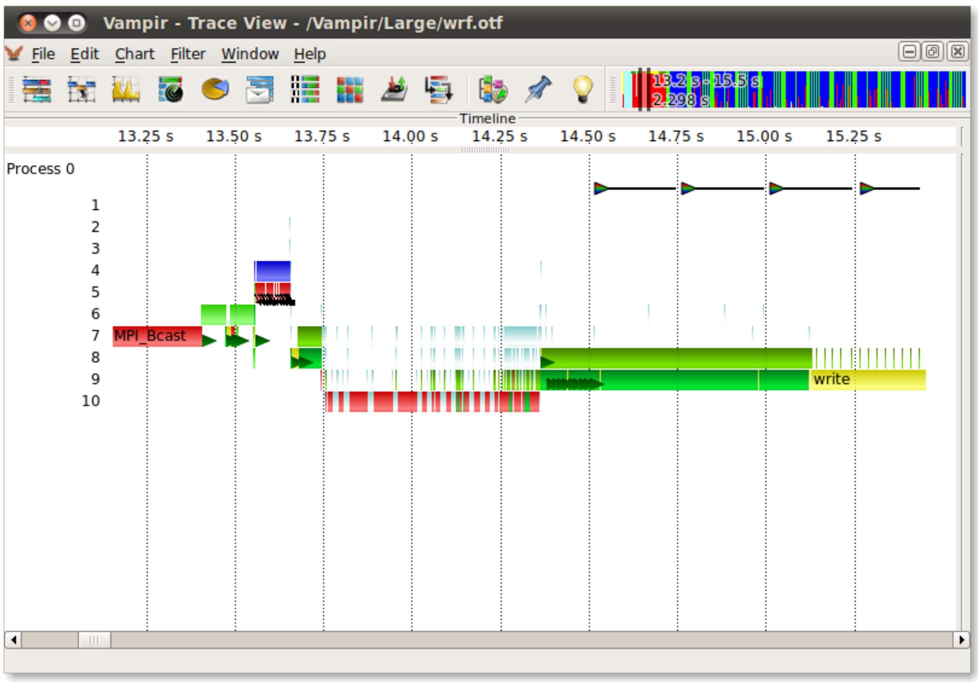


Figure 4.2: Process Timeline

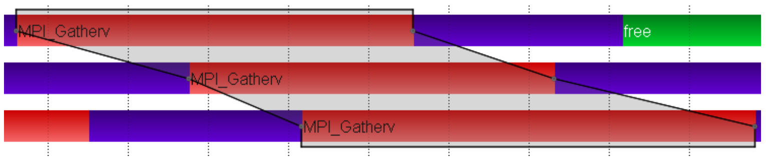


Figure 4.3: Selected MPI Collective in Master Timeline

show detailed information about that particular function, e.g., its corresponding function group name, time interval, and the complete name. The “Context View” display is explained in Chapter 4.3.3.

Some function invocations are very short. Hence these are not shown in the overall view due to a lack of display pixels. A zooming mechanism is provided to inspect a specific time interval in more detail. For further information on zooming see Section 3.3. If zooming has been performed, scrolling in horizontal direction is possible with the mouse wheel or the scroll bar at the bottom.

The “Process Timeline” resembles the “Master Timeline” with slight differences. The chart’s timeline is divided into levels, which represent the different call stack levels of function calls. The initial function begins at the first level, a sub-function called by that function is located a level beneath and so forth. If a sub-function returns to its caller, the graphical representation also returns to the level above.

In addition to the display of categorized function invocations, *Vampir*’s “Master-” and “Process Timeline” also provide information about communication events. Messages exchanged between two different processes are depicted as black lines. In timeline charts, the progress in time is reproduced from left to right. The leftmost (starting) point of a message line and its underlying process bar therefore identify the sender of the message, whereas the rightmost position of the same line represents the receiver of the message. The corresponding function calls usually reflect a pair of MPI communication directives like *MPI_Send()* and *MPI_Recv()*. Collective communication like *MPI_Gatherv()* is also displayed in the “Master Timeline” as shown in Figure 4.3.

Furthermore, additional information like message bursts, markers and I/O events is available. Table 4.1 shows the symbols and descriptions of these objects.

Since the “Process Timeline” reveals information of one process only, short black arrows are used to indicate outgoing communication. Clicking on message lines or arrows shows message details like sender process, receiver process, message length, message duration, and message tag in the “Context View” display.

The “Master Timeline” also provides the possibility to search for function and function group occurrences. In order to activate the search mode use the context menu and select “Find...”. After activation an input field appears at the top of the “Master Timeline”.

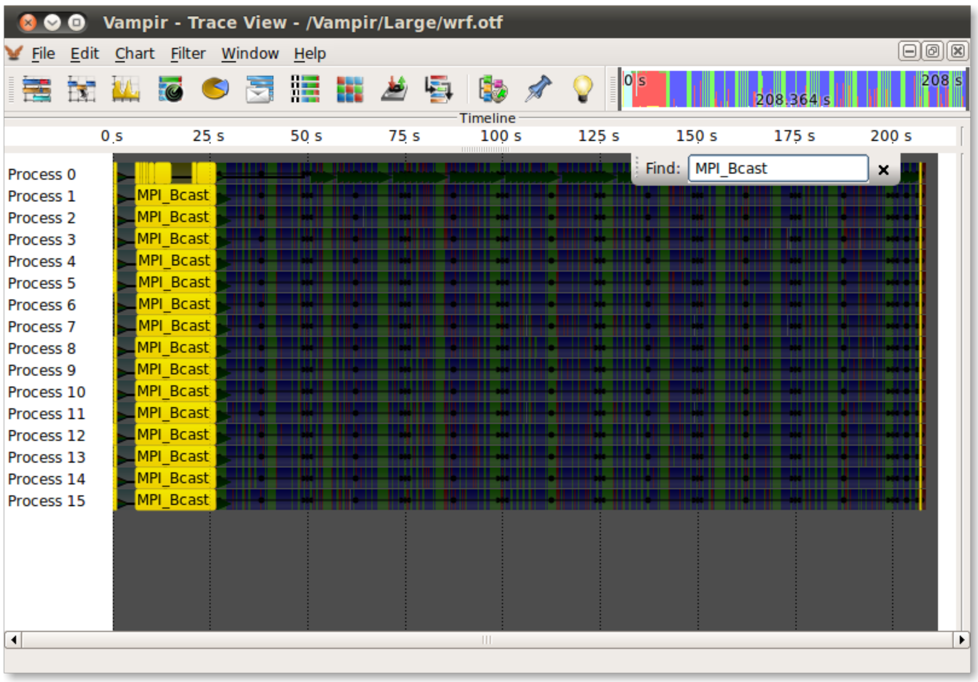


Figure 4.4: Search for MPI_Bcast in the Master Timeline

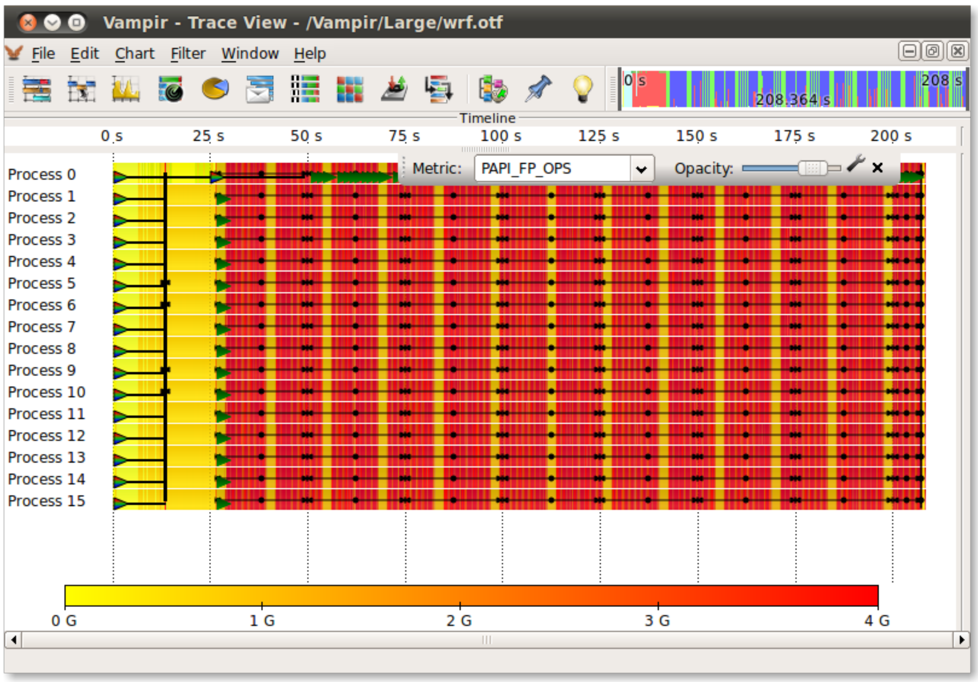


Figure 4.5: Active overlay showing PAPI_FP_OPS in the Master Timeline

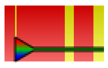
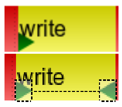
Symbol	Description
Message Burst ⊢•⊣	Due to a lack of pixels it is not possible to display a large amount of messages in a very short time interval. Therefore outgoing messages are summarized as so-called message bursts. In this representation you cannot determine which processes receive these messages. Zooming into this interval reveals the corresponding single messages.
Markers ▼ multiple ▼ single	To indicate particular points of interest during the runtime of an application, like errors or warnings, markers can be placed in a trace file. They are drawn as triangles which are colored according to their types. To indicate that two or more markers are located at the same pixel, a tricolored triangle is drawn.
I/O Events  	*Vampir* shows detailed information about I/O operations, if they are included in the trace file. I/O events are depicted as triangles at the beginning of an I/O interval. In order to see the whole interval of a single I/O event its triangle has to be selected. In that case a second triangle indicating the end of the interval appears. Multiple I/O events are tricolored and drawn as a triangle with a line to the end of the interval.

Table 4.1: Additional Information in the Master and Process Timeline

A search string can be written in this field and all corresponding function and function group occurrences are highlighted in yellow in the "Master Timeline". An example search for the function *MPI_Bcast* is depicted in Figure 4.4.

Furthermore, the "Master Timeline" also features an overlay mode for performance counter data, Figure 4.5. In order to activate the overlay mode use the context menu "Options → Performance Data". When the overlay mode is active a control window appears at the top of "Master Timeline". It allows to select the displayed counter data (metric). The counter data is displayed in a color coded fashion like in the "Performance Radar", Section 4.1.3. The color scale can be freely customized by clicking on the wrench icon. The control window also provides an opacity control slider. This slider allows to adjust the opacity of the overlay and thus makes the underlying functions easily visible without the need to disable the overlay mode.

4.1.2 Counter Data Timeline

Counters are values collected over time to count certain events like floating point operations or cache misses. Counter values can be used to store not just hardware performance counters but arbitrary sample values. There can be counters for different statistical information as well, for instance counting the number of function calls or a value in an iterative approximation of the final result. Counters are defined during the instrumentation of the application and can be individually assigned to processes.

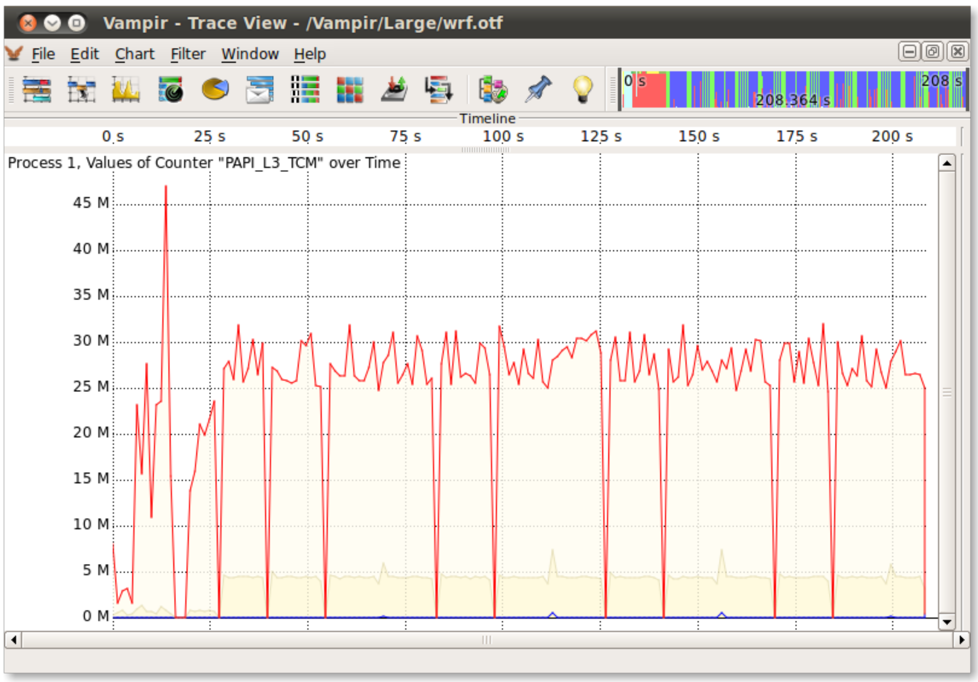


Figure 4.6: Counter Data Timeline

An example “Counter Data Timeline” chart is shown in Figure 4.6. The chart is restricted to one counter at a time. It shows the selected counter for one measuring point (e.g., process). Using multiple instances of the “Counter Data Timeline”, counters or processes can be compared easily.

The displayed graph in the chart is constructed from actual measurements (data points). Since display space is limited it is likely that there are more data points than display pixels available. In that case multiple data points need to be displayed on one pixel (width). Therefore the counter values are displayed in three graphs. A maximum line (red), an average line (yellow), and a minimum line (blue). When multiple data points need to be displayed on one pixel width, the red line shows the data point with the highest value, the blue line shows the point with the lowest value, and the yellow line indicates the average of all data points lying on this pixel width. When zooming into an smaller time range less data points need to be displayed on the available pixel space.

Eventually, when zooming far enough only one data point needs to be display on one pixel. Then also the three graphs will merge together. The actual measured data points can be displayed in the chart by enabling them via the context menu under “Options...”.

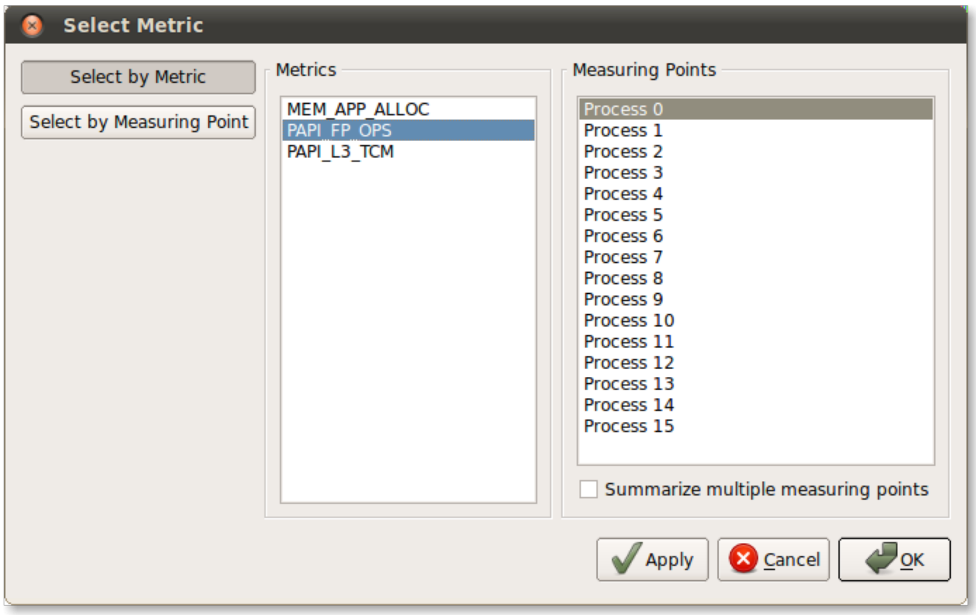
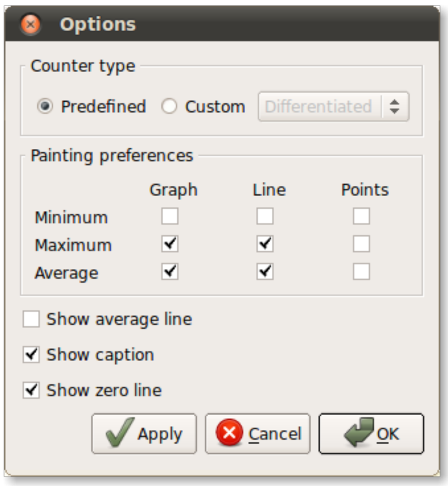


Figure 4.7: Select metric dialog

The context menu entry “Select Metric...” opens the selection dialog depicted in Figure 4.7. This dialog allows to choose the displayed counter in the chart. Each counter is defined by its metric and its measuring point. Note, depending on the measurement not all metrics might be available on all measurement points.
The two left buttons in the dialog decide whether the counter should be selected by metric or by measuring point first. In the case of “Select by Metric” there is also the option to “Summarize multiple measuring points” available. This option allows to identify outlier by summarizing counters (e.g., PAPI_FP_OPS) over multiple measuring points (e.g., processes). Hence, when this option is active multiple measuring points can be selected. The counter for the selected metric is then summarized over all selected measuring points. The displayed counter graphs in the chart need then to be read as follows. The yellow “average” line in the middle displays the average value (e.g., PAPI_FP_OPS) of all selected measuring points (e.g., processes) at a given time. The red “maximum” line shows the highest value that one of the selected measuring points achieved at a given time. The blue “minimum” line shows the smallest value that one of the selected measuring points (e.g., process) achieved at a given time. A click with the left mouse button on any point in the chart reveals its details in the “Context View” display. Stated are the min, max, and average values and the measurement points (e.g., processes) that achieved maximum and minimum values at the selected point in time.

The options dialog is depicted in Figure 4.8. It can be enabled via the context menu



Counter type

[✓ Predefined] Custom Differentiated

Painting preferences

	Graph	Line	Points
Minimum	☐	☐	☐
Maximum	✓	✓	☐
Average	✓	✓	☐

☐ Show average line

✓ Show caption

✓ Show zero line

Figure 4.8: Counter Timeline options dialog

under “Options...”. It provides the possibility include/exclude the graphs, lines and data points from the chart. It is also possible to enable an average line showing the average value of all data points in the visible area.
The counter type setting is used to determine how the data points should be connected. This is dependent on the type of the counter and usually predefined during the recording of the trace data. Nevertheless, this setting can also be changed afterwards in *Vampir*.

The “Counter Data Timeline” chart allows to create custom metrics. This process is described in Section 4.1.3. Created custom metrics become available in the “Select Metric” dialog.

4.1.3 Performance Radar

The “Performance Radar” chart, Figure 4.9, displays counter data and provides the possibility to create custom metrics. In contrast to the “Counter Data Timeline” the “Performance Radar” shows one counter for all processes at once. The values of the counter are displayed in a color coded fashion.

The displayed counter in the chart can be chosen via the context menu entry “Set Metric”. Own created custom metrics are listed under this option as well.

The option “Adjust Bar Height to” allows to change the height of the displayed value bars in the chart. This useful for traces with a high number of processes. Here the option “Adjust Bar Height to → Fit Chart Height” tries to display all processes in the chart. This enables a overview of the counter data across the entire application run.

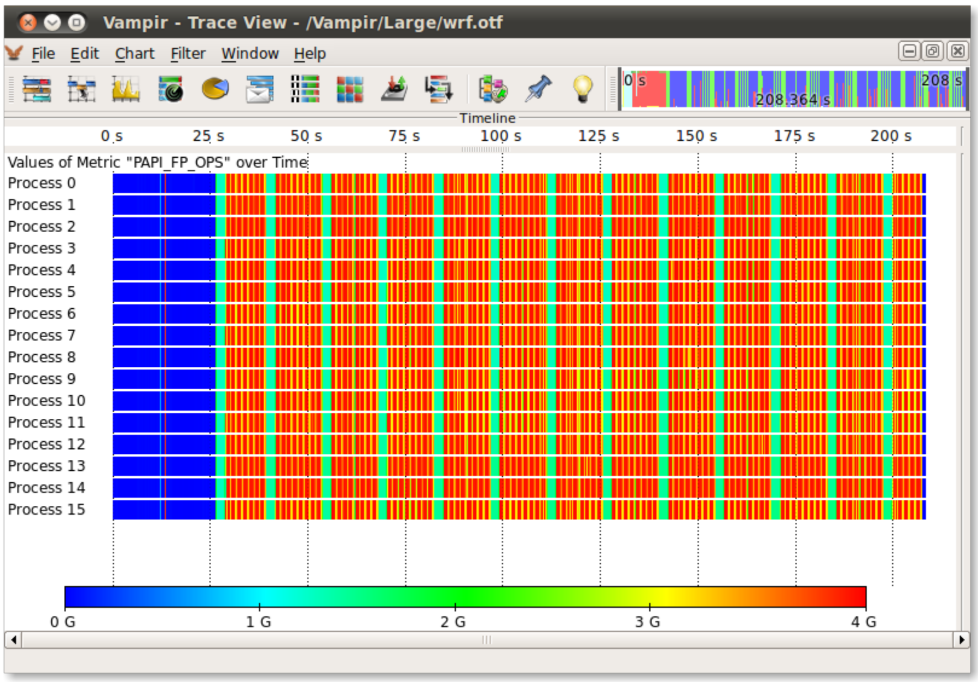


Figure 4.9: Performance Radar

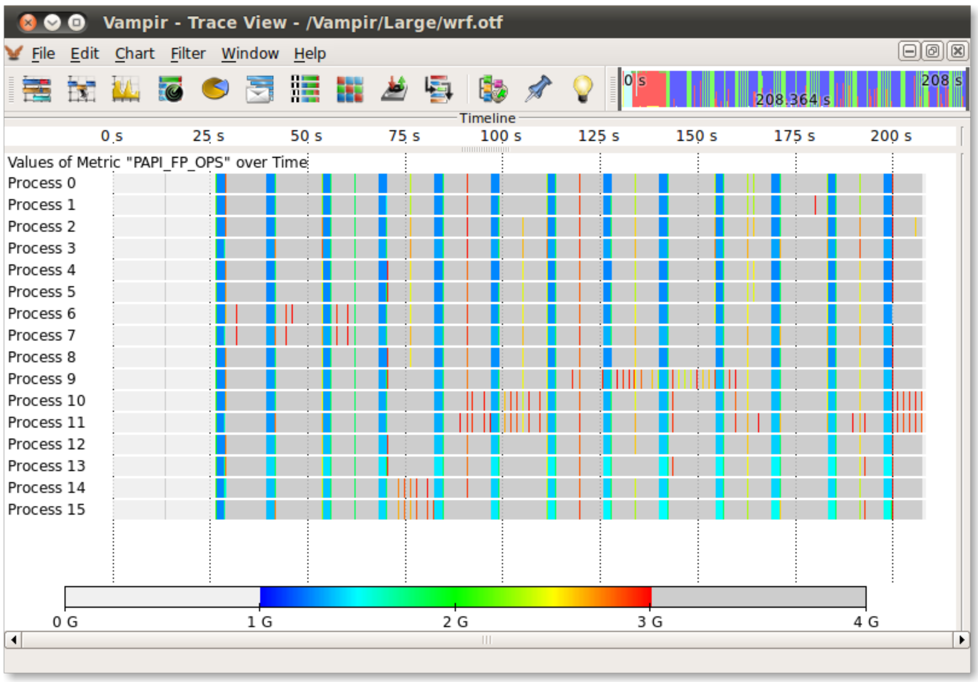
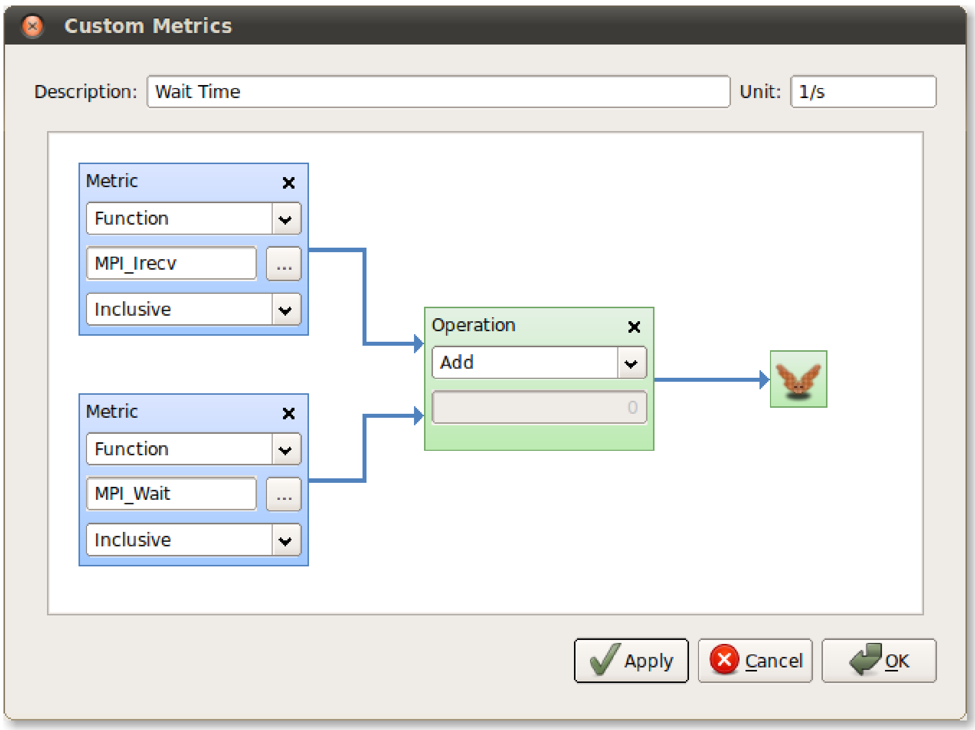


Figure 4.10: Adjusted value range in color scale

“Set Chart Mode” allows to define whether minimum, maximum, or average values should be shown. This setting comes into effect when multiple measured data points need to be displayed on one pixel. If “Maximum” or “Minimum” is active, the data point with the highest or lowest value is displayed, respectively. In case of “Average” the average of all data points on the respective pixel width is displayed. This process is also explained in the section “Counter Data Timeline” 4.1.2.

The value range of the color scale can be easily adjusted with the left mouse button. In order to adjust the color coded value rage just drag the edges of the color scale to the desired positions. Figure 4.10 shows the “Performance Radar” chart in Figure 4.9 with a smaller value range of 1G - 3G FLOPS. This allows to easily spot areas of high or low performance in the trace file. The selected value range can also be dragged to other positions in the color scale. A double-click with the left mouse button on the color scale resets the selected value range.
The option “Options → Color Scale...” in the context menu of the chart allows to customize the color scale to the own preferences.


Custom Metrics

Description: Wait Time
Unit: 1/s

Metric: Function, MPI_Irecv, Inclusive
Metric: Function, MPI_Wait, Inclusive
Operation: Add, 0

Figure 4.11: Custom Metrics Editor

The “Custom Metrics Editor” allows to derive own metrics based on existing counters and functions. The editor is accessible via the context menu entry “Customize Metrics...”. Figure 4.11 shows an example custom metric *Wait Time*. This metric is an addition of the time spent in the functions *MPI_Irecv* and *MPI_Wait*. Custom Metrics are build from input metrics that are liked together with the available operations. The context menu accessible via the right mouse button allows to add new input metrics and operations. All created custom metrics become available in the “Set Metric...” selections of the “Performance Radar” and “Counter Data Timeline” charts. There are available as well in the overlay mode of the “Master Timeline”. Custom metrics can be exported and imported in order to use them in multiple trace files.

4.2 Statistical Charts

4.2.1 Function Summary

The “Function Summary” chart, Figure 4.12, gives an overview of the accumulated time consumption across all function groups and functions. For example every time a process calls the *MPI_Send()* function the elapsed time of that function is added to the *MPI* function group time. The chart gives a condensed view of the execution of the application. A comparison between the different function groups can be made and dominant function groups can be distinguished easily.

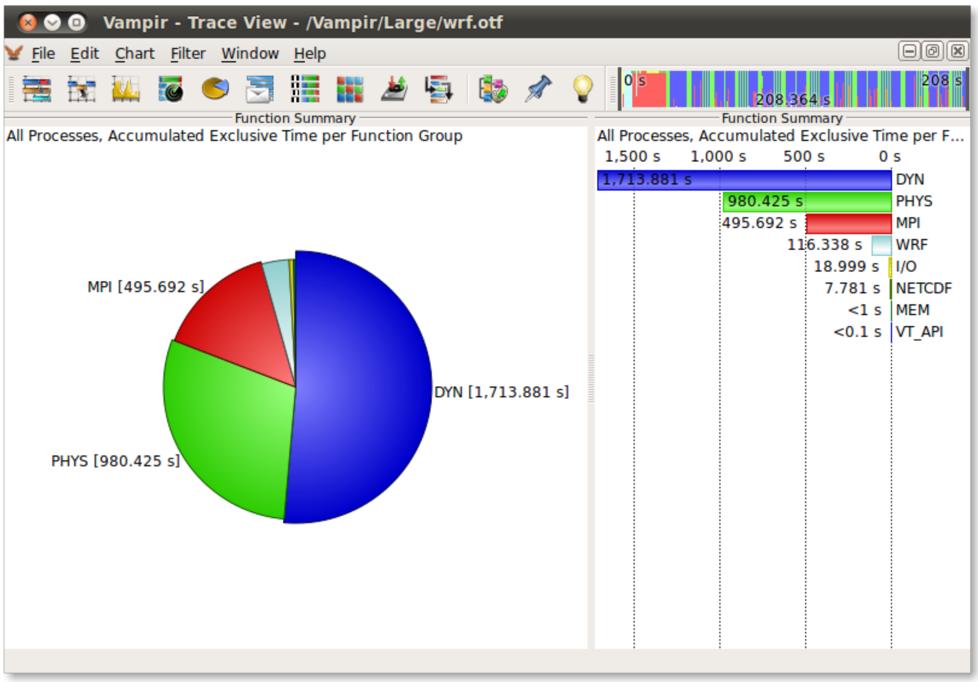


Figure 4.12: Function Summary

It is possible to change the information displayed via the context menu entry “Set Metric” that offers options like “Average Exclusive Time”, “Number of Invocations”, “Accumulated Inclusive Time”, etc.

Note: “Inclusive” means the amount of time spent in a function and all of its subroutines. “Exclusive” means the amount of time spent in just this function.

The context menu entry “Set Event Category” specifies whether function groups or functions should be displayed in the chart. The functions own the color of their function group.

It is possible to hide functions and function groups from the displayed information with the context menu entry “Filter”. In order to mark the function or function group to be filtered just click on the associated label or color representation in the chart. Using the “Process Filter” (see Section 4.4) allows you to restrict this chart to a set of processes. As a result, only the consumed time of these processes is displayed for each function group or function. Instead of using the filter which affects all other displays by hiding processes, it is possible to select a single process via “Set Process” in the context menu of the “Function Summary”. This does not have any effect on other charts.

The “Function Summary” can be shown as “Histogram” (a bar chart, like in timeline charts) or as “Pie Chart”. To switch between these representations use the “Set Chart Mode” entry of the context menu.

The shown functions or function groups can be sorted by name or value via the context menu option “Sort By”.

4.2.2 Process Summary

The “Process Summary”, depicted in Figure 4.13, is similar to the “Function Summary” but shows the information for every process independently. This is useful for analyzing the balance between processes to reveal bottlenecks. For instance finding that one process spends a significantly high time performing the calculations could indicate an unbalanced distribution of work and therefore can slow down the whole application.

The context menu entry “Set Event Category” specifies whether function groups or functions should be displayed in the chart. The functions own the color of their function group.

The chart calculates statistics based on “Number of Invocations”, “Accumulated Inclusive Time”, or “Accumulated Exclusive Time”. To change between these three modes use the context menu entry “Set Metric”.

The number of clustered profile bars is based on the chart height by default. You can also disable the clustering or set a fixed number of clusters via the context menu entry “Clustering” by selecting the corresponding value in the spin box. Located left of the clustered profile bars is a graphical overview indicating the processes associated to the cluster. Moving the cursor over the blue areas in the overview opens a tooltip stating the respective process name.

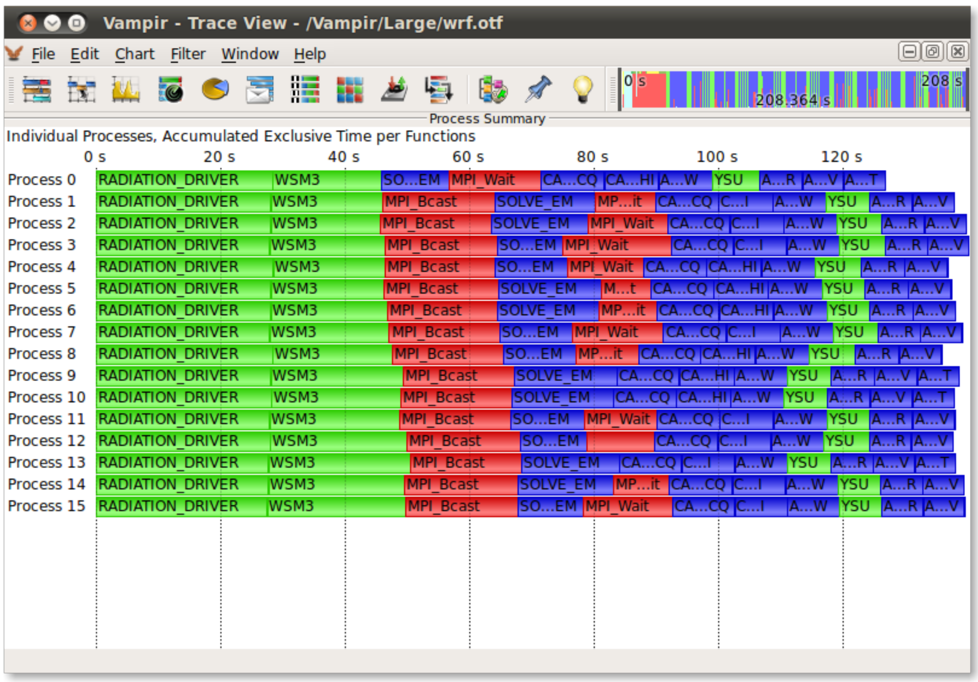


Figure 4.13: Process Summary

It is possible to profile only one function or function group or to hide arbitrary functions and function groups from the displayed information. To mark the function or function group to be profiled or filtered just click on the associated color representation in the chart. The context menu entries “Profile of Selected Function/(Group)” and “Filter Selected Function/(Group)” will then provide the possibility to profile or filter the selected function or function group. Using the “Process Filter” (see Section 4.4) allows you to restrict this view to a set of processes.

The context menu entry “Sort by” allows you to order function profiles by “Number of Clusters”. This option is only available if the chart is currently showing clusters. Otherwise function profiles are sorted automatically by process. While profiling one function the menu entry “Sort by Value” allows to order functions by their execution time.

4.2.3 Message Summary

The “Message Summary” is a statistical chart showing an overview of all messages grouped by certain characteristics, Figure 4.14.

All values are represented in a bar chart fashion. The number next to each bar is the group base while the number inside a bar depicts the values depending on the chosen metric. Therefore, the “Set Metric” sub-menu of the context menu can be used to switch

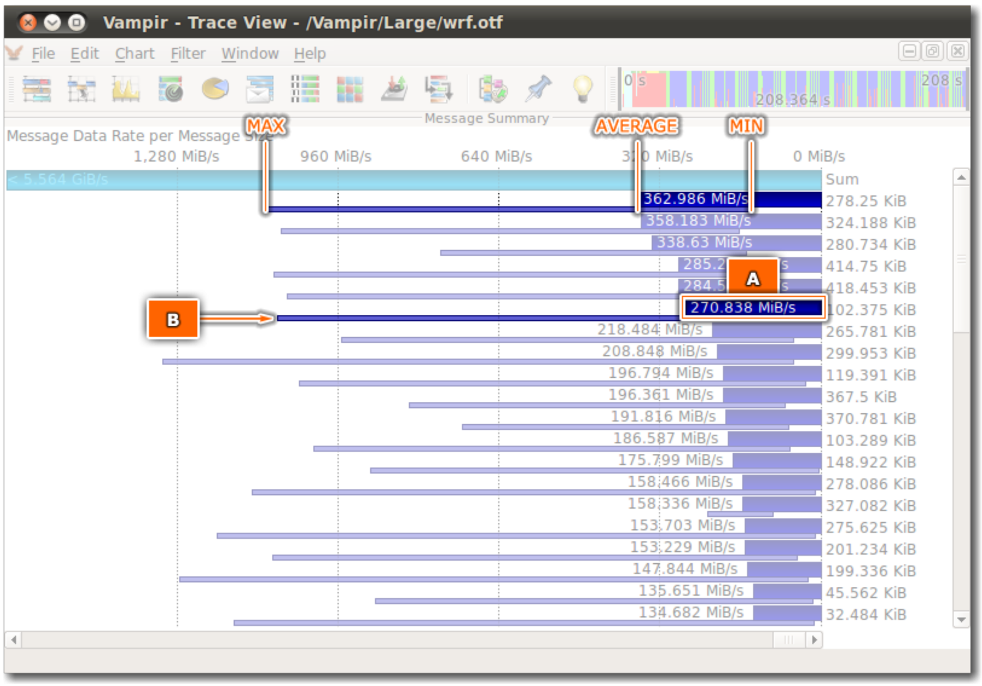


Figure 4.14: Message Summary Chart with metric set to “Message Transfer Rate” showing the average transfer rate (A), and the minimal/maximal transfer rate (B)

between “Aggregated Message Volume”, “Message Size”, “Number of Messages”, and “Message Transfer Rate”.

The group base can be selected via the context menu entry “Group By”. Possible options are “Message Size”, “Message Tag”, and “Communicator (MPI)”.

Note: There will be one bar for every occurring group. However, if the metric is set to “Message Transfer Rate”, the minimal and the maximal transfer rate is given in an additional small bar beneath the main bar showing the average transfer rate. The additional bar starts at the minimal rate and ends at the maximal rate, see Figure 4.14.

In order to filter out messages click on the associated label or color representation in the chart and then choose “Filter” from the context menu.

4.2.4 Communication Matrix View

The “Communication Matrix View” is another way of analyzing communication imbalances. It shows information about messages sent between processes.

The chart, as shown in Figure 4.15, is figured as a table. Its rows represent the sending processes whereas the columns represent the receivers. The color legend on the right

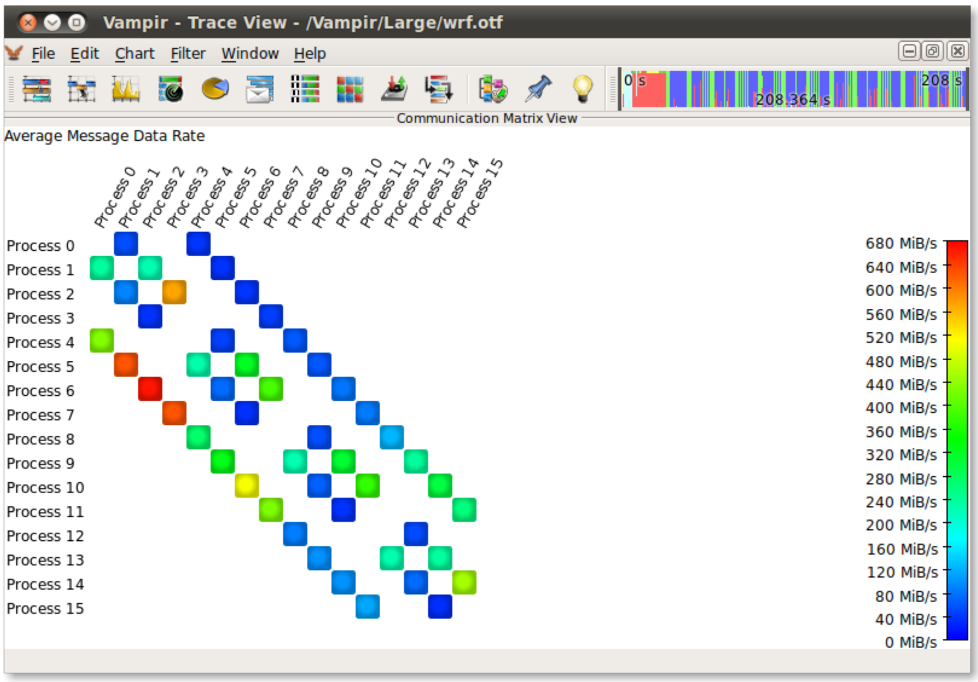


Figure 4.15: Communication Matrix View

indicates the displayed values. It adapts automatically to the currently shown value range.

It is possible to change the type of displayed values. Different metrics like the average duration of messages passed from sender to recipient or minimum and maximum bandwidth are offered. To change the type of value that is displayed use the context menu option “Set Metric”.

Use the “Process Filter” to define which processes/groups should be displayed. (see Section 4.4).

Note: A high duration is not automatically caused by a slow communication path between two processes, but can also be due to the fact that the time between starting transmission and successful reception of the message can be increased by a recipient that delays reception for some reason. This will cause the duration to increase (by this delay) and the message rate, which is the size of the message divided by the duration, to decrease accordingly.

4.2.5 I/O Summary

The “I/O Summary”, depicted in Figure 4.16, is a statistical chart giving an overview of the input-/output operations recorded in the trace file.

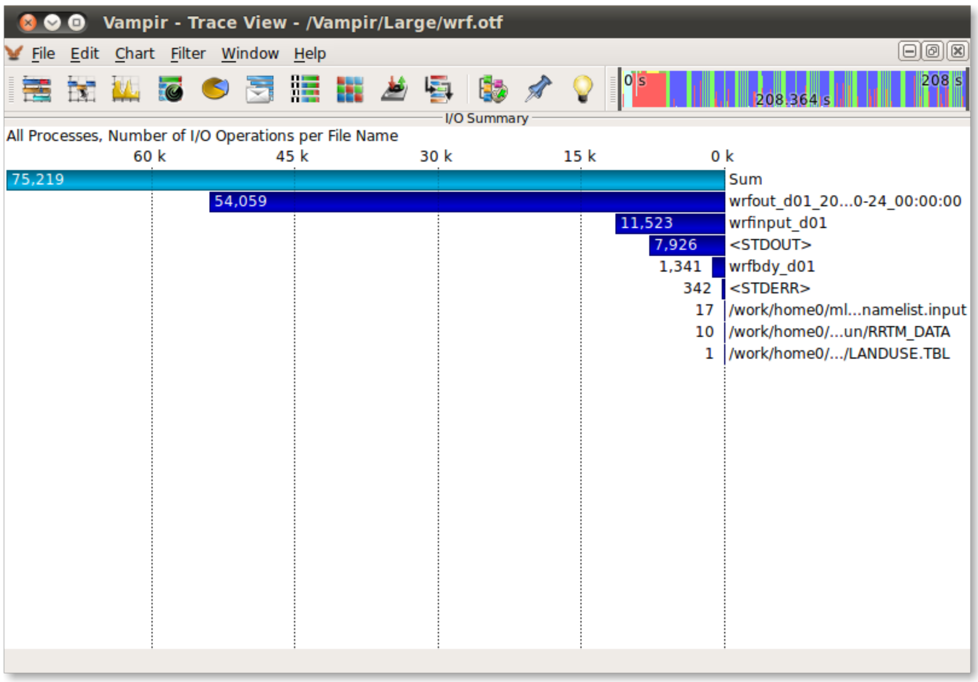

Vampir - Trace View - /Vampir/Large/wrf.otf
File Edit Chart Filter Window Help
0 s
208.364 s
208 s
I/O Summary
All Processes, Number of I/O Operations per File Name
60 k
45 k
30 k
15 k
0 k
75,219 Sum
54,059 wrfout_d01_20...0-24_00:00:00
11,523 wrfinput_d01
7,926 <STDOUT>
1,341 wrfbdy_d01
342 <STDERR>
17 /work/home0/ml...namelist.input
10 /work/home0/...un/RRTM_DATA
1 /work/home0/.../LANDUSE.TBL

Figure 4.16: I/O Summary

All values are represented in a histogram like fashion. The text label indicates the group base while the number inside each bar represents the value of the chosen metric. The “Set Metric” sub-menu of the context menu is used to switch between the available metrics “Number of I/O Operations”, “Accumulated I/O Transaction Sizes”, and all ranges of “I/O Operation Size”, “I/O Transaction Time”, or “I/O Bandwidth”.

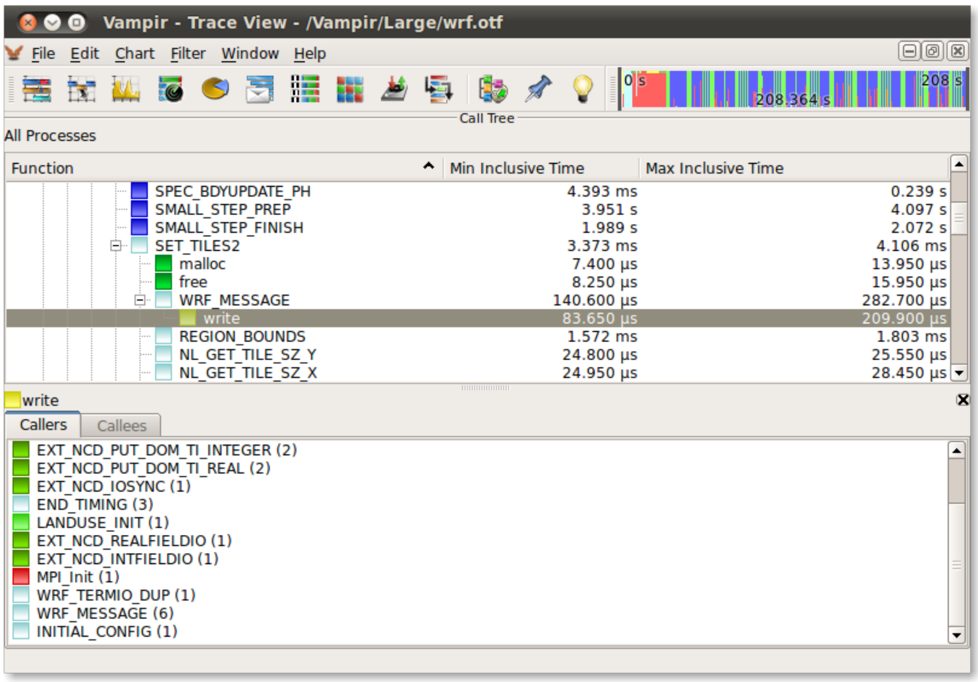
The I/O operations can be grouped by the characteristics “Transaction Size”, “File Name”, and “Operation Type”. The group base can be changed via the context menu entry “Group I/O Operations by”.

Note: There will be one bar for every occurring metric. For a quick and convenient overview it is also possible to show minimum, maximum, and average values at once. This option is available for the metrics “Transaction Size Range of I/O Operations”, “Time Range of I/O Operations”, and “Bandwidth Range of I/O Operations”. The minimum and maximum values are shown in an additional, smaller bar beneath the main bar indicating the average value. The additional bar starts at the minimum and ends at the maximum value of the metric, see Figure 4.14.

In order to select the I/O operation types that should be considered for the statistic calculation use the “Set I/O Operations” sub-menu of the context menu. Available options are “Read”, “Write”, “Read, Write”, and “Apply Global I/O Operations Filter”. The latter includes all selected operation types from the “I/O Events” filter dialog, see Chapter 4.4.

4.2.6 Call Tree

The “Call Tree”, depicted in Figure 4.17, illustrates the invocation hierarchy of all monitored functions in a tree representation. The display reveals information about the number of invocations of a given function, the time spent in the different calls and the caller-callee relationship.



Function	Min Inclusive Time	Max Inclusive Time
SPEC_BDYUPDATE_PH	4.393 ms	0.239 s
SMALL_STEP_PREP	3.951 s	4.097 s
SMALL_STEP_FINISH	1.989 s	2.072 s
SET_TILES2	3.373 ms	4.106 ms
malloc	7.400 µs	13.950 µs
free	8.250 µs	15.950 µs
WRF_MESSAGE	140.600 µs	282.700 µs
write	83.650 µs	209.900 µs
REGION_BOUNDS	1.572 ms	1.803 ms
NL_GET_TILE_SZ_Y	24.800 µs	25.550 µs
NL_GET_TILE_SZ_X	24.950 µs	28.450 µs

write

Callers | Callees

EXT_NCD_PUT_DOM_TI_INTEGER (2)
EXT_NCD_PUT_DOM_TI_REAL (2)
EXT_NCD_IOSYNC (1)
END_TIMING (3)
LANDUSE_INIT (1)
EXT_NCD_REALFIELDIO (1)
EXT_NCD_INTFIELDIO (1)
MPI_Init (1)
WRF_TERMIO_DUP (1)
WRF_MESSAGE (6)
INITIAL_CONFIG (1)

Figure 4.17: Call Tree

The entries of the “Call Tree” can be sorted in various ways. Simply click on one header of the tree representation to use its characteristic to re-sort the “Call Tree”. Please note that not all available characteristics are enabled by default. To add or remove characteristics use the “Set Metric” sub-menu of the context menu.

To leaf through the different function calls, it is possible to fold and unfold the levels of the tree. This can be achieved by double clicking a level, or by using the fold level buttons next to the function name.

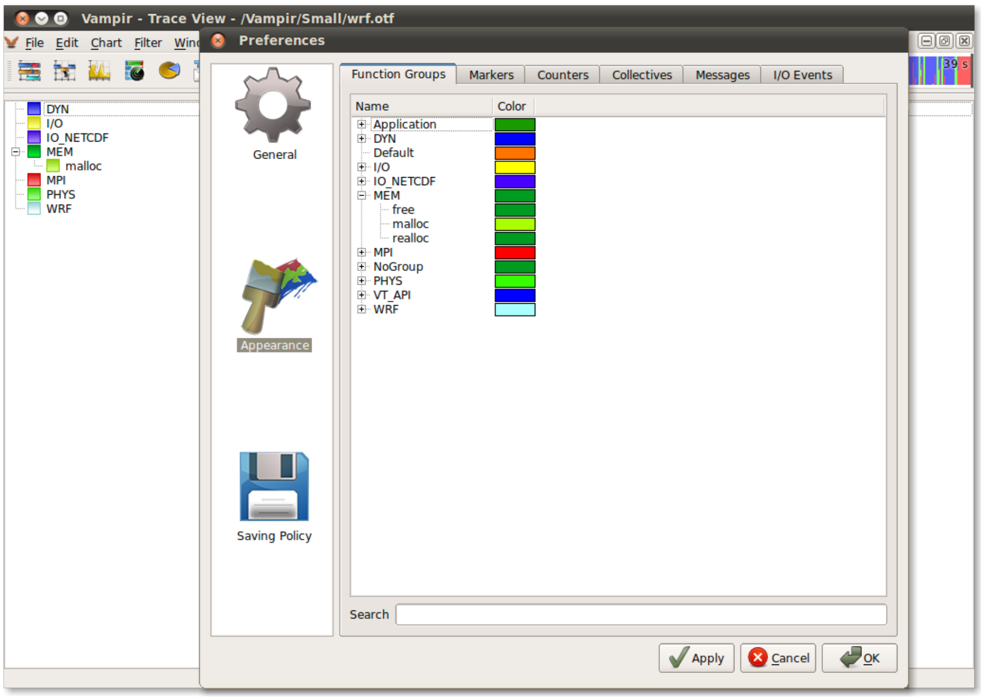
Functions can be called by many different caller functions, what is hardly obvious in the tree representation. Therefore, a relation view shows all callers and callees of the currently selected function in two separated lists, shown in the lower area in Figure 4.17.

In order to find a certain function by its name, *Vampir* provides a search option accessible via the context menu entry “Find...”. The entered keyword has to be confirmed by pressing the *Return* key. The “Previous” and “Next” buttons can be used to flip through the results.

4.3 Informational Charts

4.3.1 Function Legend

The “Function Legend” lists all visible function groups of the loaded trace file along with their corresponding color.



- 🟦 DYN
- 🟨 I/O
- 🟪 IO_NETCDF
- 🟩 MEM
  - 🟩 malloc
- 🟥 MPI
- 🟩 PHYS
- ⬜ WRF

Name	Color
Application	🟩
DYN	🟦
Default	🟧
I/O	🟨
IO_NETCDF	🟪
MEM	🟩
free	🟩
malloc	🟩
realloc	🟩
MPI	🟥
NoGroup	🟩
PHYS	🟩
VT_API	🟦
WRF	⬜

Figure 4.18: Function Legend

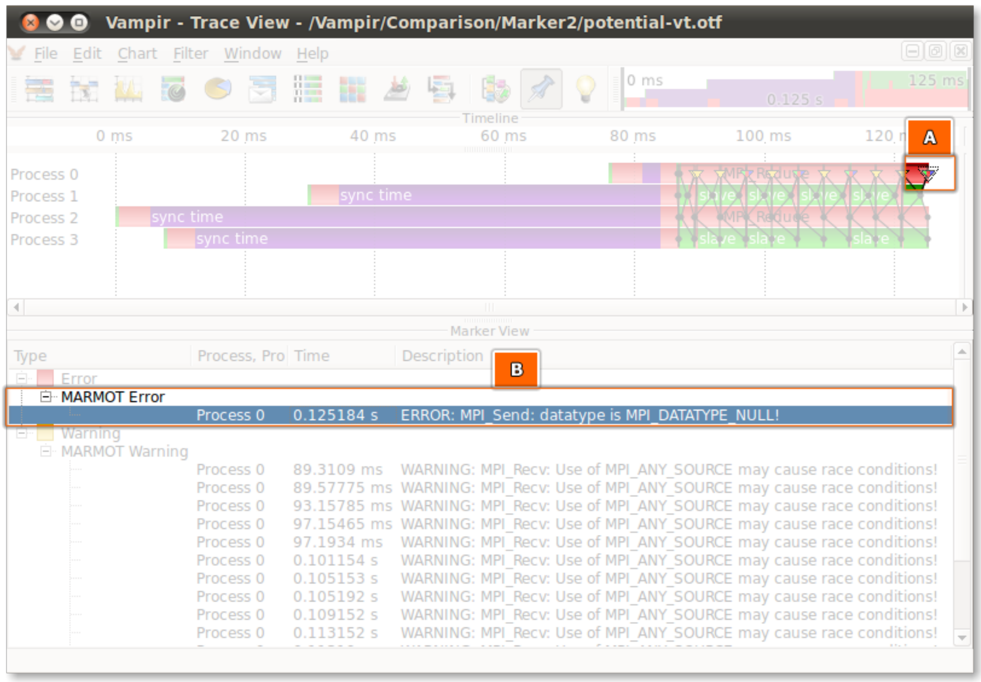
If colors of functions are changed, they appear in a tree like fashion under their respective function group as well, see Figure 4.18.

4.3.2 Marker View

The “Marker View” lists all marker events included in the trace file.

The display organizes the marker events based on their respective groups and types in a tree like fashion. Additional information like the time of occurrence or descriptions are provided for each marker.

By clicking on a marker event in the “Marker View” this event becomes selected in the timeline displays. If this marker is located outside the visible area the zoom jumps to this event automatically. It is possible to select marker events by their type as well.



Type	Process, Pro	Time	Description
Error			
MARMOT Error			
	Process 0	0.125184 s	ERROR: MPI_Send: datatype is MPI_DATATYPE_NULL!
Warning			
MARMOT Warning			
	Process 0	89.3109 ms	WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
	Process 0	89.57775 ms	WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
	Process 0	93.15785 ms	WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
	Process 0	97.15465 ms	WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
	Process 0	97.1934 ms	WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
	Process 0	0.101154 s	WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
	Process 0	0.105153 s	WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
	Process 0	0.105192 s	WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
	Process 0	0.109152 s	WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!
	Process 0	0.113152 s	WARNING: MPI_Recv: Use of MPI_ANY_SOURCE may cause race conditions!

Figure 4.19: A chosen marker (A) and its representation in the Marker View (B)

Then all events belonging to that type are selected in the “Master Timeline” and the “Process Timeline”. By holding the “Ctrl” or “Shift” key pressed multiple marker events can be selected. If exactly two marker events are selected the zoom is set automatically to the occurrence time of the markers.

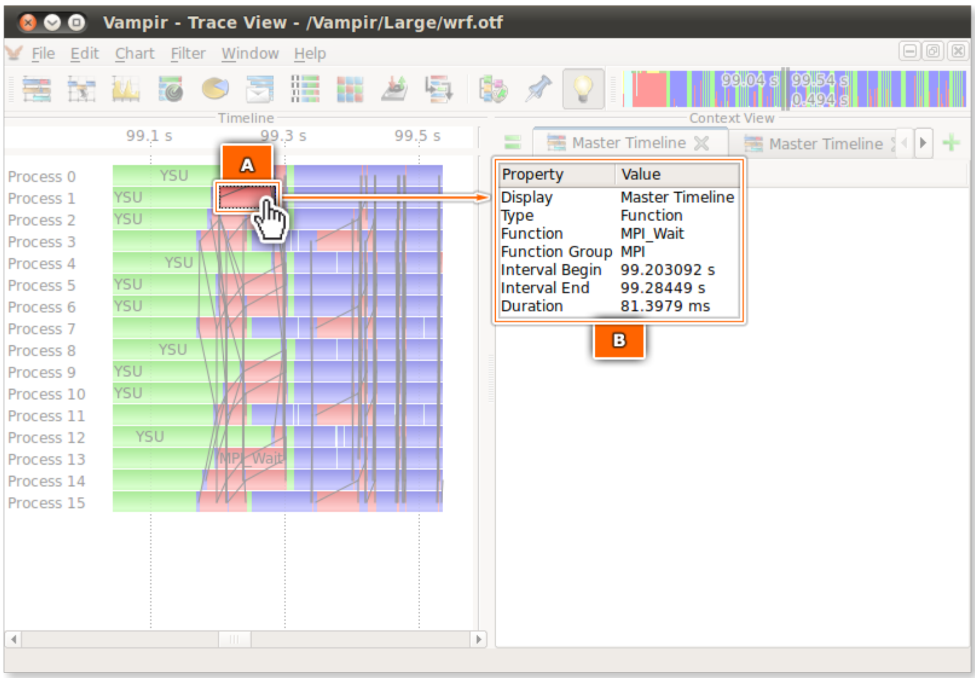
4.3.3 Context View

As implied by its name, the “Context View” provides more detailed information of a selected object compared to its graphical representation.

An object, e.g., a function, function group, message, or message burst can be selected directly in a chart by clicking its graphical representation. For different types of objects different context information is provided by the “Context View”. For example the object specific information for functions includes properties like “Interval Begin”, “Interval End”, and “Duration”, shown in Figure 4.20.

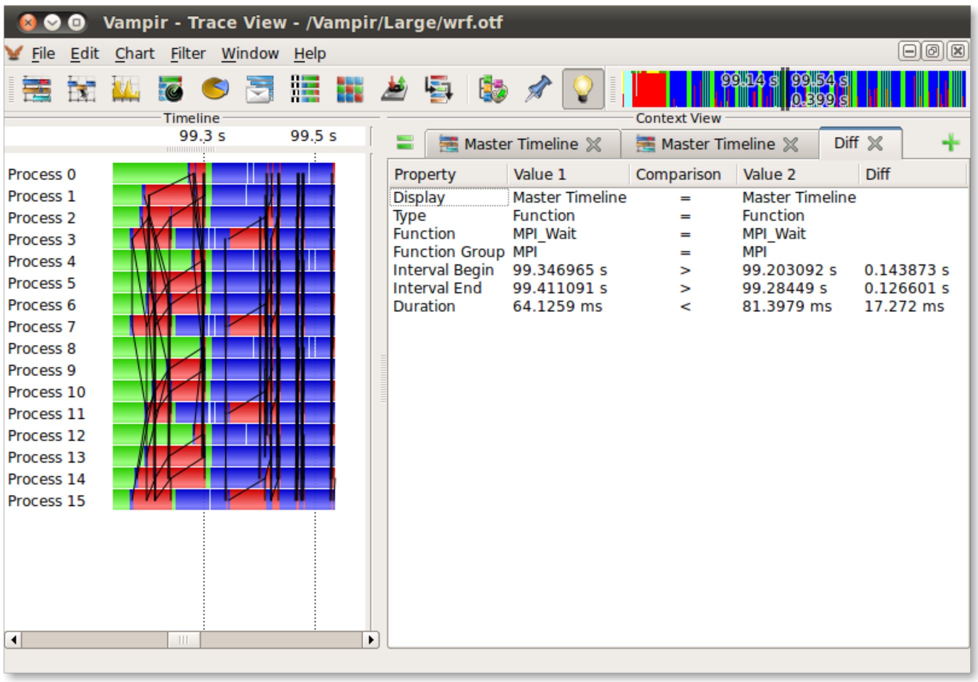
The “Context View” may contain several tabs. A new empty one can be added by clicking the “+”-symbol on the right hand side. Information of new selected objects are always displayed in the currently active tab.

The “Context View” offers a mode for the comparison of information between tabs. Just use the “=”-button on the left hand side and choose two objects to compare. It is possible to compare different objects from different charts. This can be useful in some



Property	Value
Display	Master Timeline
Type	Function
Function	MPI_Wait
Function Group	MPI
Interval Begin	99.203092 s
Interval End	99.28449 s
Duration	81.3979 ms

Figure 4.20: Context View, showing context information (B) of a selected function (A)



Property	Value 1	Comparison	Value 2	Diff
Display	Master Timeline	=	Master Timeline	
Type	Function	=	Function	
Function	MPI_Wait	=	MPI_Wait	
Function Group	MPI	=	MPI	
Interval Begin	99.346965 s	>	99.203092 s	0.143873 s
Interval End	99.411091 s	>	99.28449 s	0.126601 s
Duration	64.1259 ms	<	81.3979 ms	17.272 ms

Figure 4.21: Comparison between Context Information

cases. The comparison shows a list of common properties. The corresponding values as well as the differences are displayed. The first line always indicates the names of the respective charts, Figure 4.21.

4.4 Information Filtering and Reduction

Due to the large amount of information that can be stored in trace files, it is usually necessary to reduce the displayed information according to some filter criteria. In *Vampir*, there are different ways of filtering. It is possible to limit the displayed information to a certain choice of processes or to specific types of communication events, e.g. to certain types of messages or collective operations. Deselecting an item in a filter means that this item is fully masked. In *Vampir*, filters are global. Therefore, masked items will no longer show up in any chart. Filtering not only affects all performance charts, but also the ‘Zoom Toolbar”. The available filter can be reached via the “Filter” entry in the main menu.

Example: Figure 4.22 shows a typical process representation in the “Process Filter” dialog. This kind of representation is equal to all other filter dialog windows. Processes can be filtered by their “Process Group”, “Communicators”, and “Process Hierarchy”. Items to be filtered are arranged in a spreadsheet representation. In addition to selecting or deselecting an entire group of processes, it is also possible to filter single processes.

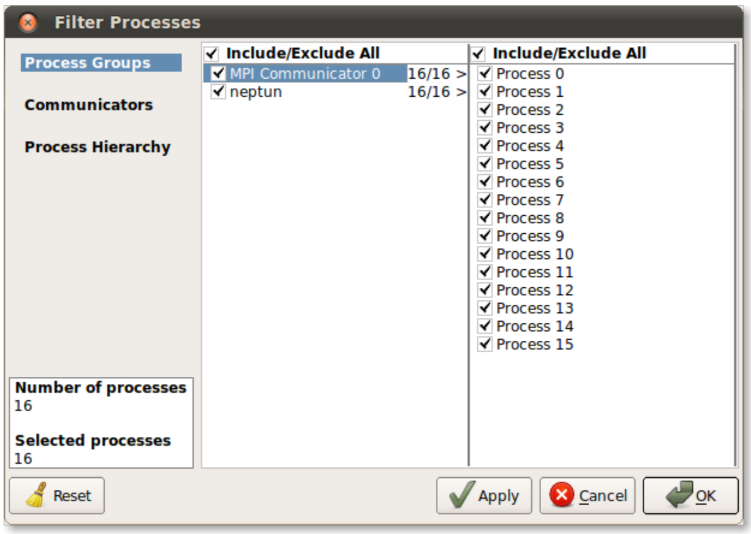


Figure 4.22: Process Filter

Different selection methods can be used in a filter. The check box “Include/Exclude All” either selects or deselects every item. Specific items can be selected/deselected by clicking into the check box next to it. Furthermore, it is possible to select/deselect multiple items at once. Therefore, mark the desired entries by clicking their names while holding either the “Shift” or the “Ctrl” key. By holding the “Shift” key every item in between the two clicked items will be marked. Holding the “Ctrl” key, on the other hand, enables you to add or remove specific items from/to the marked ones. Clicking into the check box of one of the marked entries will cause selection/deselection for all of them.

Filter Object	**Filter Criteria**
Processes	Process Groups Communicators Process Hierarchy
Collective Operations	Communicators Collective Operations
Messages	Message Communicators Message Tags
I/O Events	I/O Groups File Names Operation Types

Table 4.2: Options of Filtering

4.5 Function Filtering

The filtering of functions in *Vampir* is controlled via the “Function Filter Dialog”, depicted in Figure 4.23. This dialog can be accessed via the main menu under “Filter $\rightarrow$ Functions...”.

The filter can be enabled/disabled using the checkbox left hand side in the header (“Show only functions that...”) of the dialog.

The “Function Filter Dialog” is build on the concept of filter rules. The user can define several individual rules. The rules are explained in more detail in Chapter 4.5.1. The header of the dialog defines how multiple rules are evaluated. One possibility is to build up the filter in a way that combines the filter rules with an *and* relation. To choose this mode “all” must be selected in the combo box in the header of the dialog. This means that all rules must be true in order to produce the filter output. The other option is to combine the rules with an *or* relation. To choose this mode “any” must be selected in the combo box in the header of the dialog. In this case each rule is applied individually to the trace file. The examples in Chapter 4.5.2 illustrate both modes.

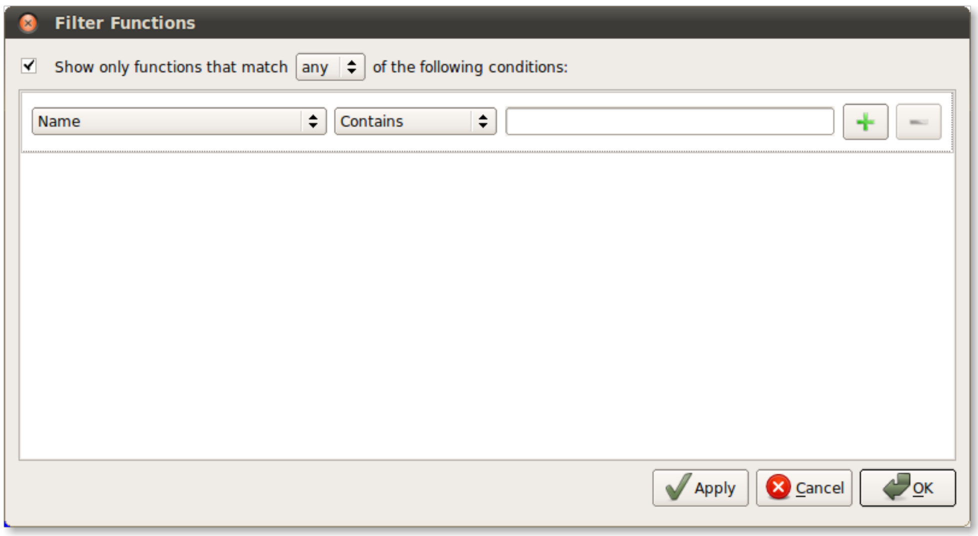


Figure 4.23: Function Filter Dialog

4.5.1 Filter Options

This chapter explains the various options available to build up filter rules.

Filtering Functions by Name

One way of filtering functions is by their name. This filter mode provides two different options.

Name provides a text field for an input string. Depending on the options, all functions whose names match the input string are shown. The matching is not case sensitive.

Available options:

- **Contains:** The given input string must occur in the function name.
- **Does not contain:** The given input string must not occur in the function name.
- **Is equal to:** The given input string must be the same as the function name.
- **Is not equal to:** The given input string must not be the same as the function name.
- **Begins with:** The function name must start with the given input string.
- **Ends with:** The function name must end with the given input string.

List of Names provides a dialog that allows to directly select the desired set of functions and function groups.

Available options:

- **Contains:** The selected functions are shown.
- **Does not contain:** The selected functions are filtered.

Filtering Functions by Duration

Functions can also be filtered by their duration. Duration of a function refers to the time spent in this function from the entry to the exit of the function.

There are two options available:

- **Is greater than:** All functions whose duration time is longer than the specified time are shown.
- **Is less than:** All functions whose duration time is shorter than the specified time are shown.

Filtering Functions by Number of Invocations

The number of invocations of a function can also be used as filter rule. This criteria refers to how often a function is executed in an application. There are two possible filter rules in this mode.

Number of Invocations shows functions based on their total number of invocations in the whole application run.

There are two options available:

- **Is greater than:** All functions whose number of invocations is greater than the specified number are shown.
- **Is less than:** All functions whose number of invocations is less than the specified number are shown.

Number of Invocations per Process shows functions based on their individual number of invocations per process. Hence, if the number of invocations of a function varies over different processes, this function might be shown for some processes and filtered for others.

There are two options available:

- **Is greater than:** All functions whose number of invocations is greater than the specified number are shown.

- **Is less than:** All functions whose number of invocations is less than the specified number are shown.

4.5.2 Examples

In this chapter a few examples explain the usage of the function filter. This enables the user to understand the basic principles of function filtering in *Vampir* “at a glace”. It also illustrates a part of the set of available filter options provided by *Vampir*.

Unfiltered Trace File

This section introduces the example trace file in an unfiltered state. The timelines show a part of the initialization of the WRF weather forecast code. The red color corresponds to communication (MPI), whereas the purple areas represent some input functions of the weather model.

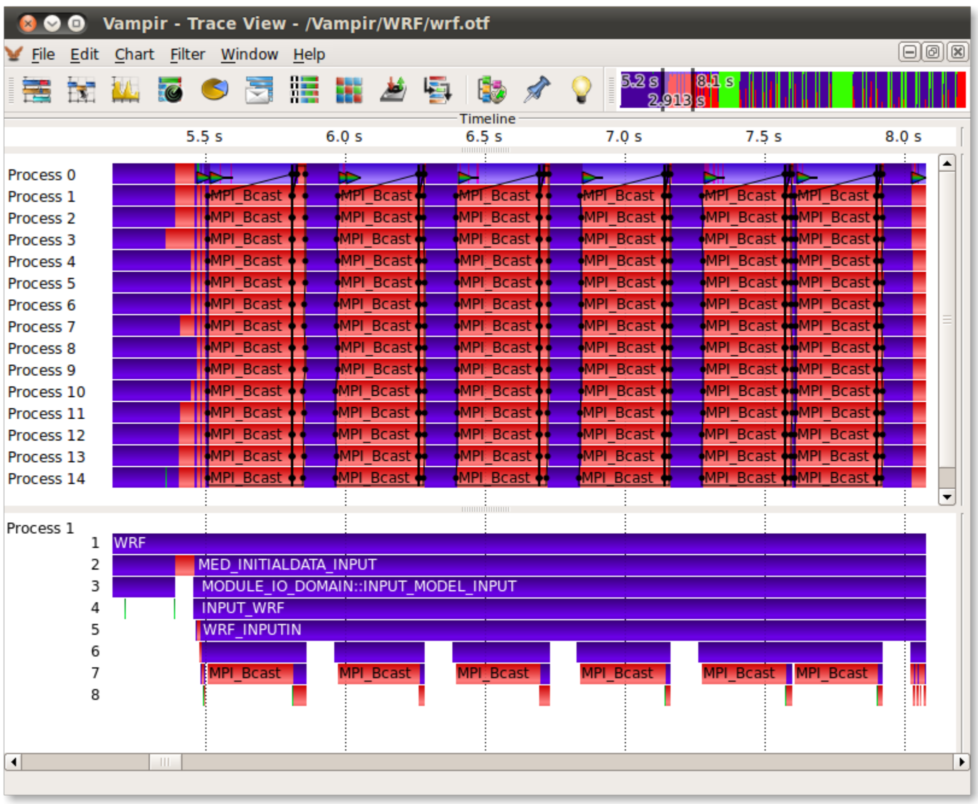
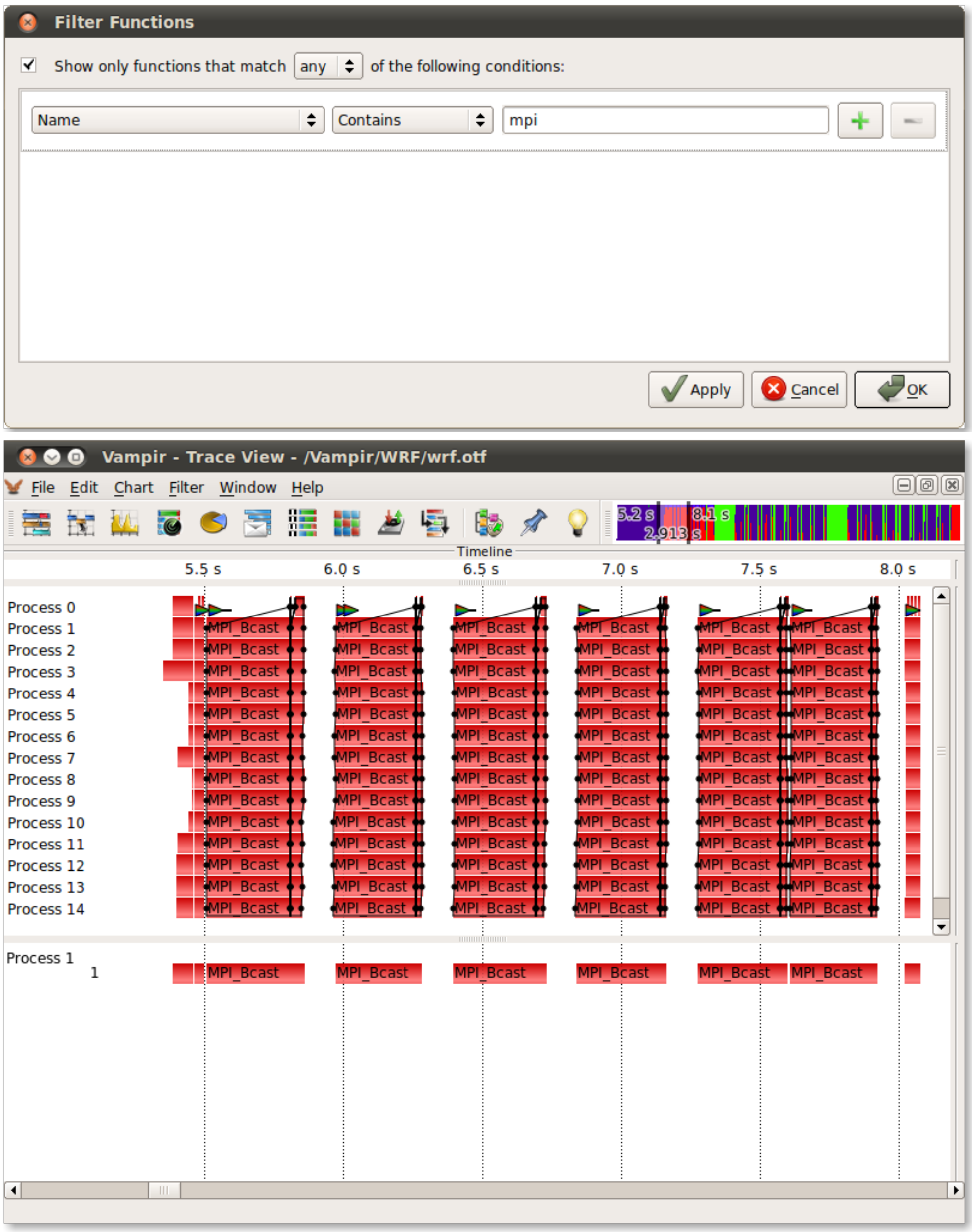


Figure 4.24: Master Timeline and Process Timeline without filtering

Showing only MPI Functions

In this example only functions that contain the string “mpi” (not case sensitive) somewhere in their name are shown. Since only MPI functions start with “MPI” in their name this filter setting shows all MPI functions and filters the others.



Show only functions that match any of the following conditions:

Name | Contains | mpi

Figure 4.25: Showing only MPI

Showing only Functions with at least 250 ms Duration

This example demonstrates the filtering of functions by their duration. Here only long function occurrences with a minimum duration time of 250 ms are shown. All other functions are filtered.

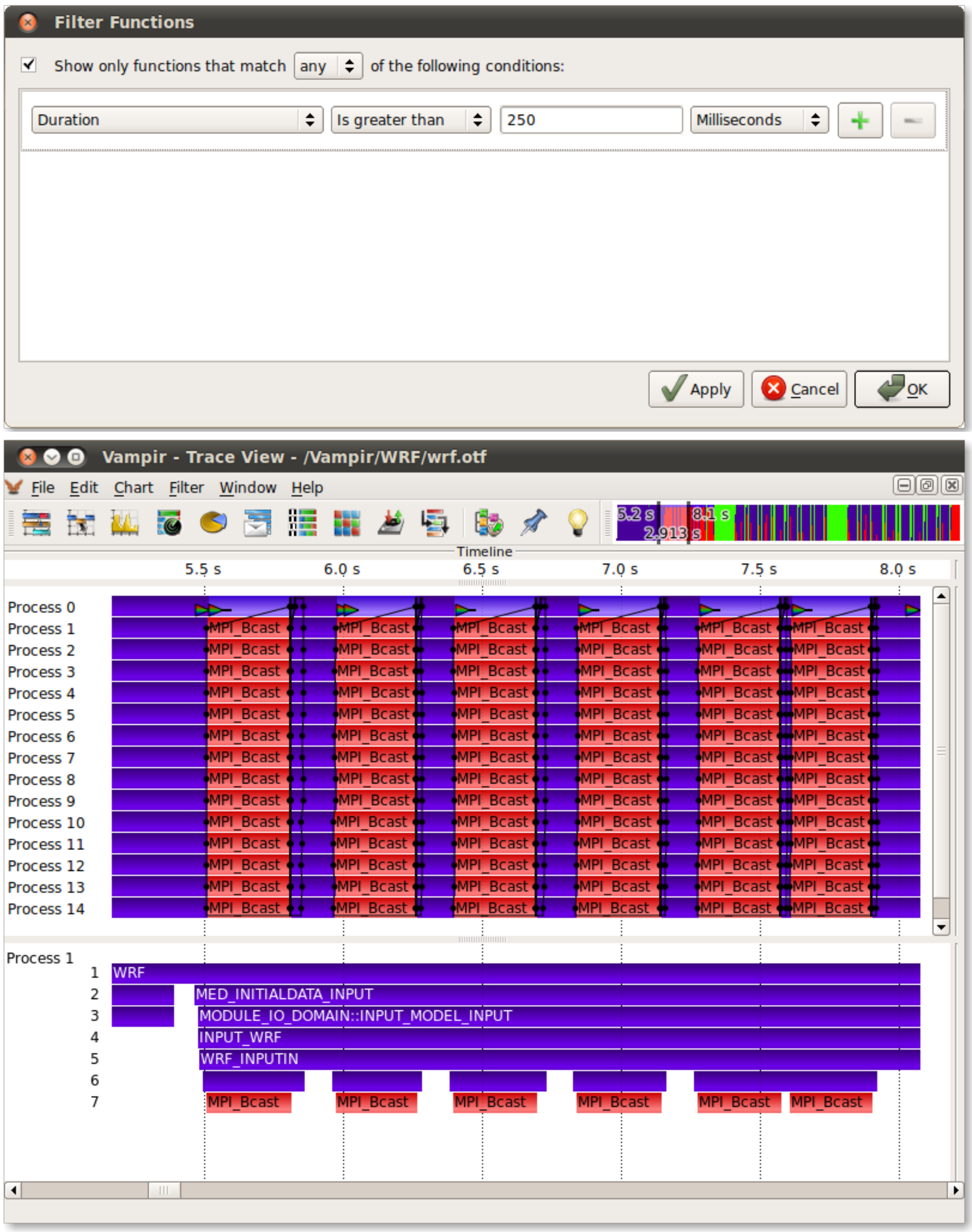
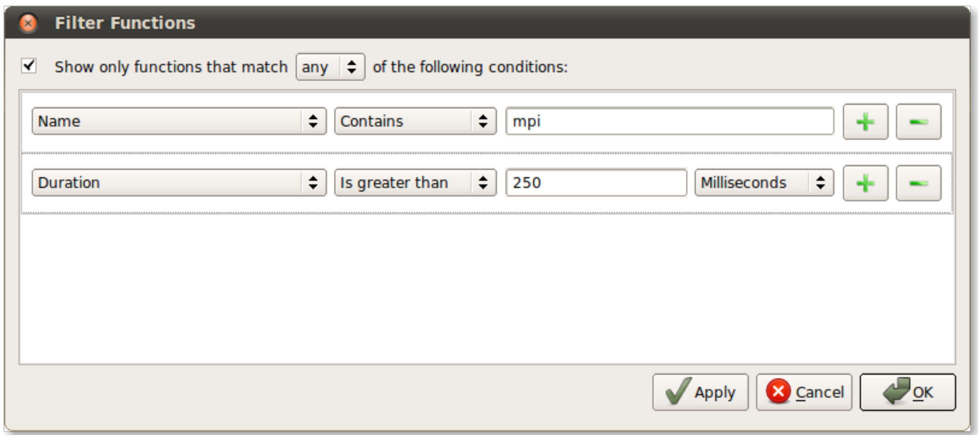


Figure 4.26: Showing only functions with more than 250 ms duration

Combining Function Name and Duration Rules

This example combines the two previous rules. First the “any” relation is used. Thus, the filter shows all functions that have at least 250 ms duration time and additionally also all “MPI” functions.



Filter Functions

☑ Show only functions that match [any] of the following conditions:

Name	Contains	mpi	
Duration	Is greater than	250	Milliseconds

Apply | Cancel | OK

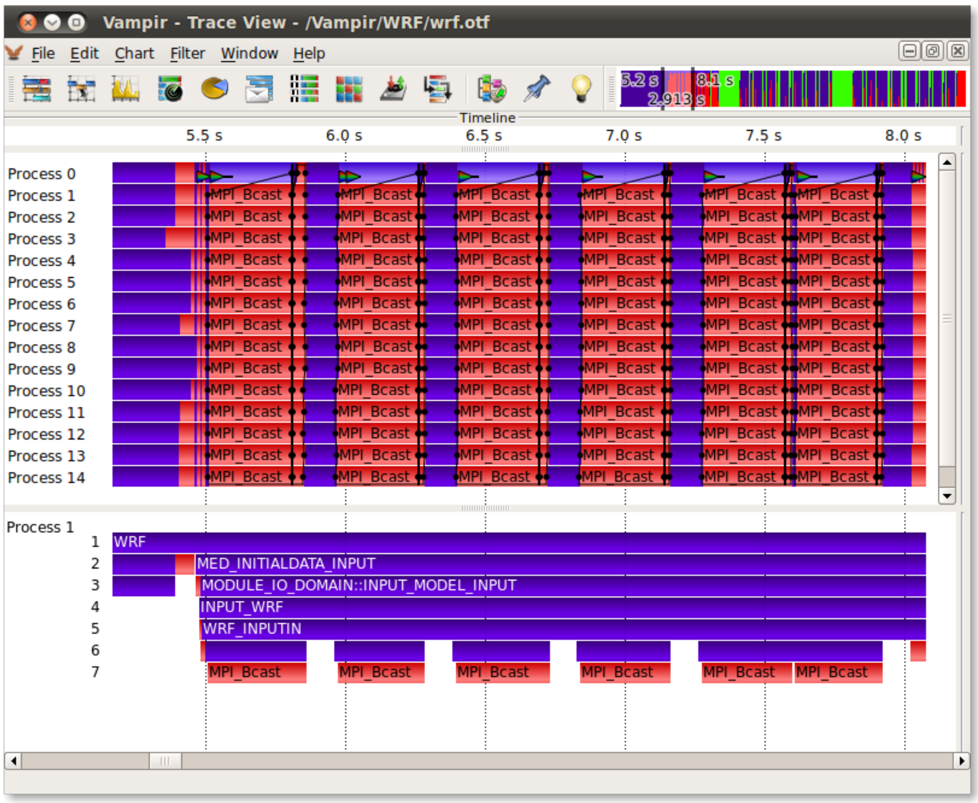
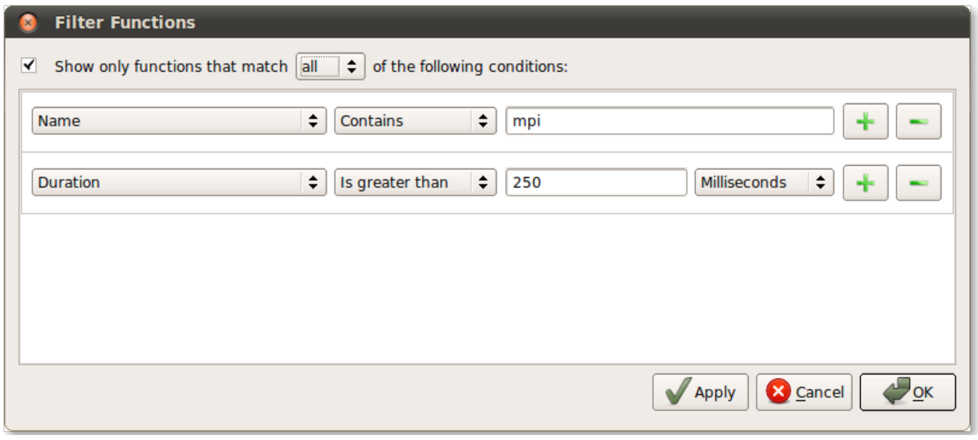


Figure 4.27: Combining rules using “any”

The second example illustrates the usage of the “all” relation. Here all shown functions have to satisfy both rules. Therefore the filter shows only “MPI” functions that have a duration time of more than 250 ms.


Filter Functions

Show only functions that match: all, of the following conditions:

Name	Contains	mpi	
Duration	Is greater than	250	Milliseconds

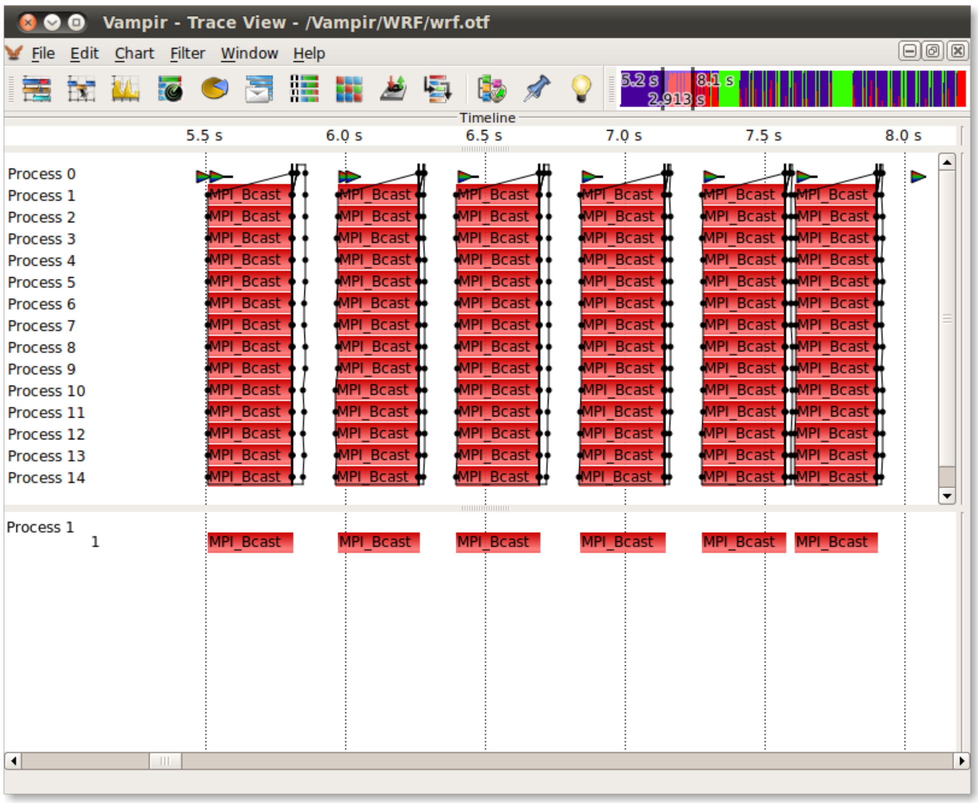
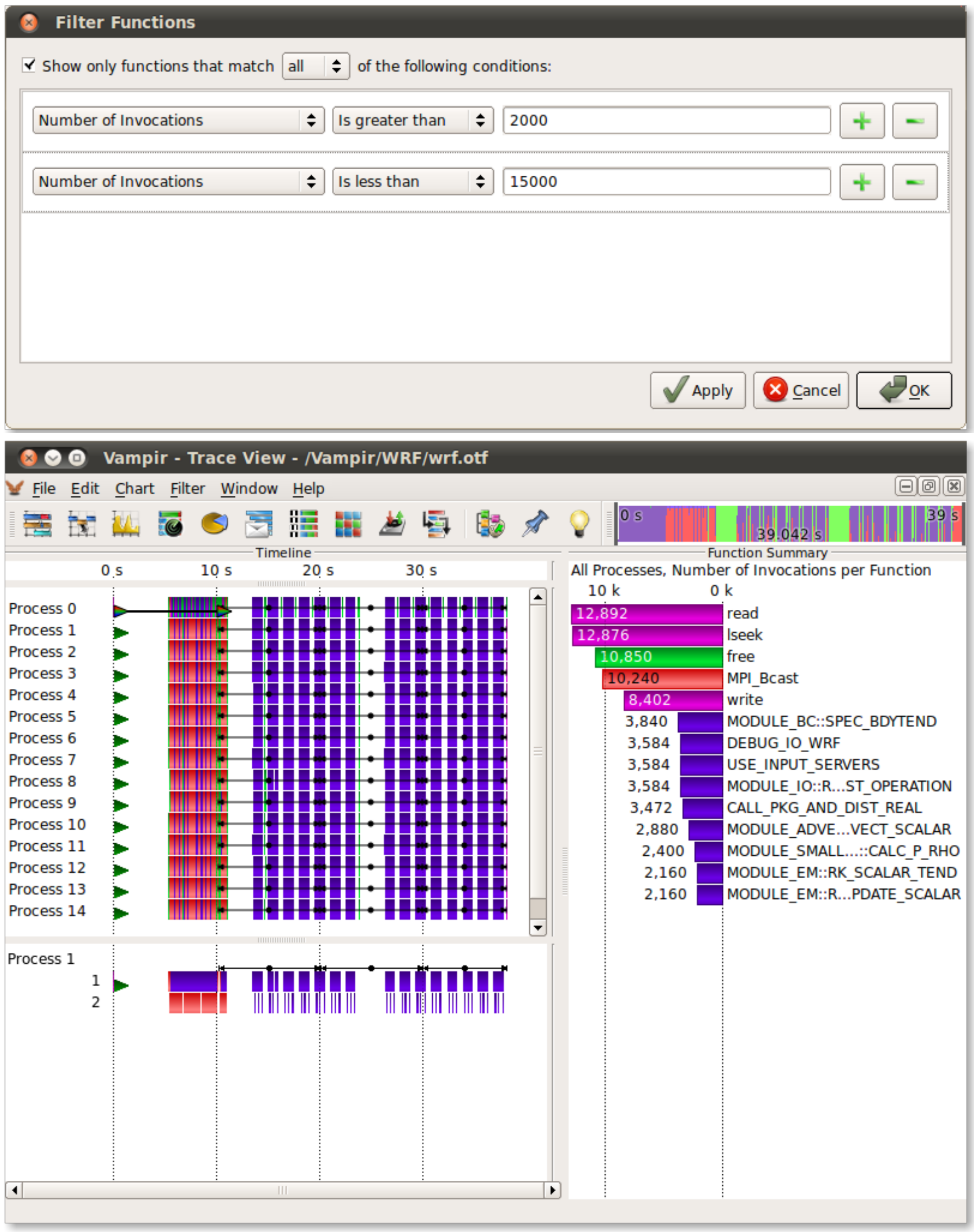


Figure 4.28: Combining rules using “all”

Building Ranges with Number of Invocation Rules

The combination of rules also allows for the filtering of functions in a specified criteria range. The following example filter setup shows all functions whose number of invocations lie inside the range between 2000 and 15000.



Filter Functions

☑ Show only functions that match all of the following conditions:

Number of Invocations	Is greater than	2000
Number of Invocations	Is less than	15000

Apply Cancel OK

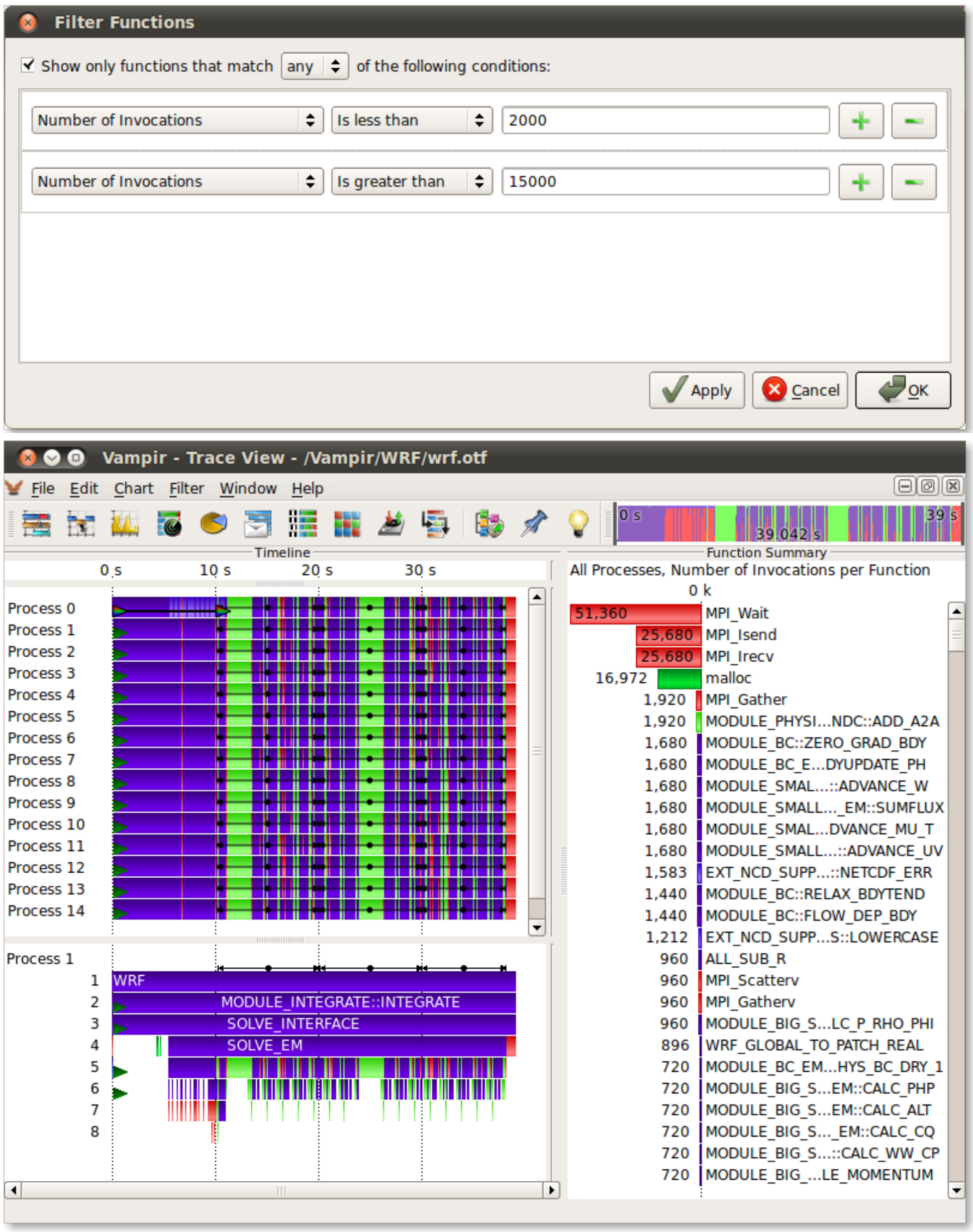
Vampir - Trace View - /Vampir/WRF/wrf.otf

Function Summary
All Processes, Number of Invocations per Function

12,892	read
12,876	lseek
10,850	free
10,240	MPI_Bcast
8,402	write
3,840	MODULE_BC::SPEC_BDYTEND
3,584	DEBUG_IO_WRF
3,584	USE_INPUT_SERVERS
3,584	MODULE_IO::R...ST_OPERATION
3,472	CALL_PKG_AND_DIST_REAL
2,880	MODULE_ADVE...VECT_SCALAR
2,400	MODULE_SMALL...::CALC_P_RHO
2,160	MODULE_EM::RK_SCALAR_TEND
2,160	MODULE_EM::R...PDATE_SCALAR

Figure 4.29: Show functions inside a specified range

This example demonstrates the opposite behavior of the previous example. Here all functions whose number of invocations lie outside the range between 2000 and 15000 are shown, i.e., functions with less than 2000 invocations and functions with more than 15000 invocations.



Filter Functions

☑ Show only functions that match any of the following conditions:

Number of Invocations | Is less than | 2000

Number of Invocations | Is greater than | 15000

Vampir - Trace View - /Vampir/WRF/wrf.otf

Function Summary — All Processes, Number of Invocations per Function

51,360	MPI_Wait
25,680	MPI_Isend
25,680	MPI_Irecv
16,972	malloc
1,920	MPI_Gather
1,920	MODULE_PHYSI...NDC::ADD_A2A
1,680	MODULE_BC::ZERO_GRAD_BDY
1,680	MODULE_BC_E...DYUPDATE_PH
1,680	MODULE_SMAL...::ADVANCE_W
1,680	MODULE_SMALL..._EM::SUMFLUX
1,680	MODULE_SMAL...DVANCE_MU_T
1,680	MODULE_SMALL...::ADVANCE_UV
1,583	EXT_NCD_SUPP...::NETCDF_ERR
1,440	MODULE_BC::RELAX_BDYTEND
1,440	MODULE_BC::FLOW_DEP_BDY
1,212	EXT_NCD_SUPP...S::LOWERCASE
960	ALL_SUB_R
960	MPI_Scatterv
960	MPI_Gatherv
960	MODULE_BIG_S...LC_P_RHO_PHI
896	WRF_GLOBAL_TO_PATCH_REAL
720	MODULE_BC_EM...HYS_BC_DRY_1
720	MODULE_BIG_S...EM::CALC_PHP
720	MODULE_BIG_S...EM::CALC_ALT
720	MODULE_BIG_S..._EM::CALC_CQ
720	MODULE_BIG_S...::CALC_WW_CP
720	MODULE_BIG_...LE_MOMENTUM

Figure 4.30: Show functions outside a specified range

5 Comparison of Trace Files

The comparison of trace files in *Vampir* extends existing functionality. That way the user can best benefit from already gained analysis experience. For the comparison of performance characteristics all common charts are provided. In order to effectively compare multiple trace files, their zoom needs to be coupled and synchronized. For comparison of selected areas of interest the trace files need to be freely shiftable in time. This allows for arbitrary alignments of the trace files, and thus, enables comparison of user selected areas in the trace data.

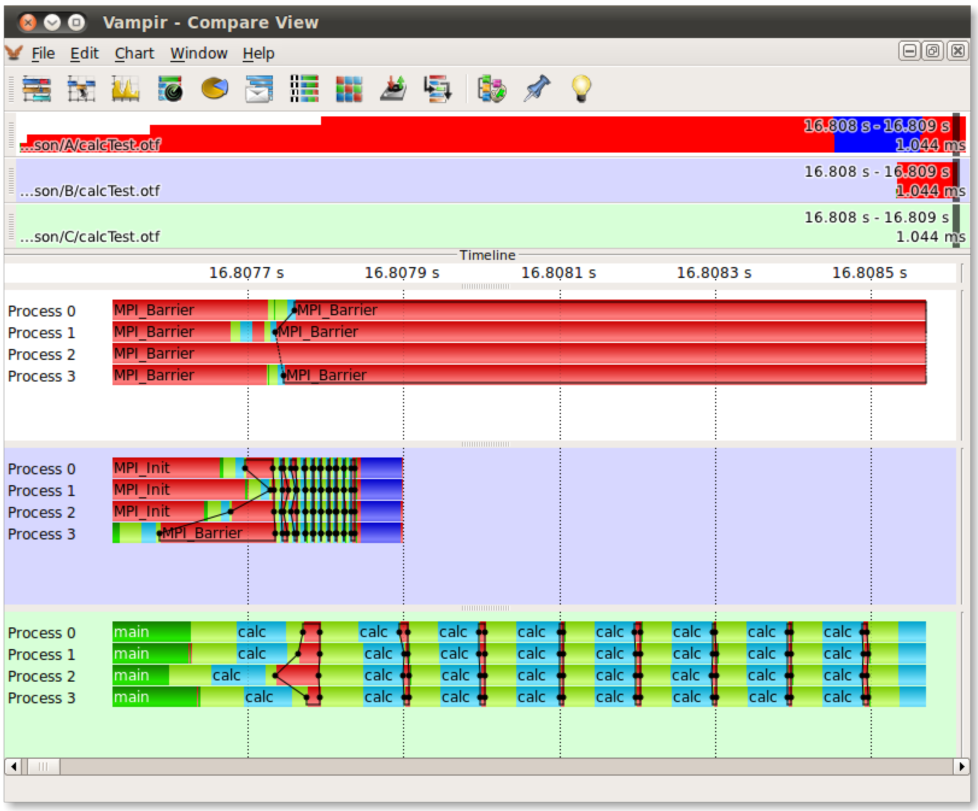


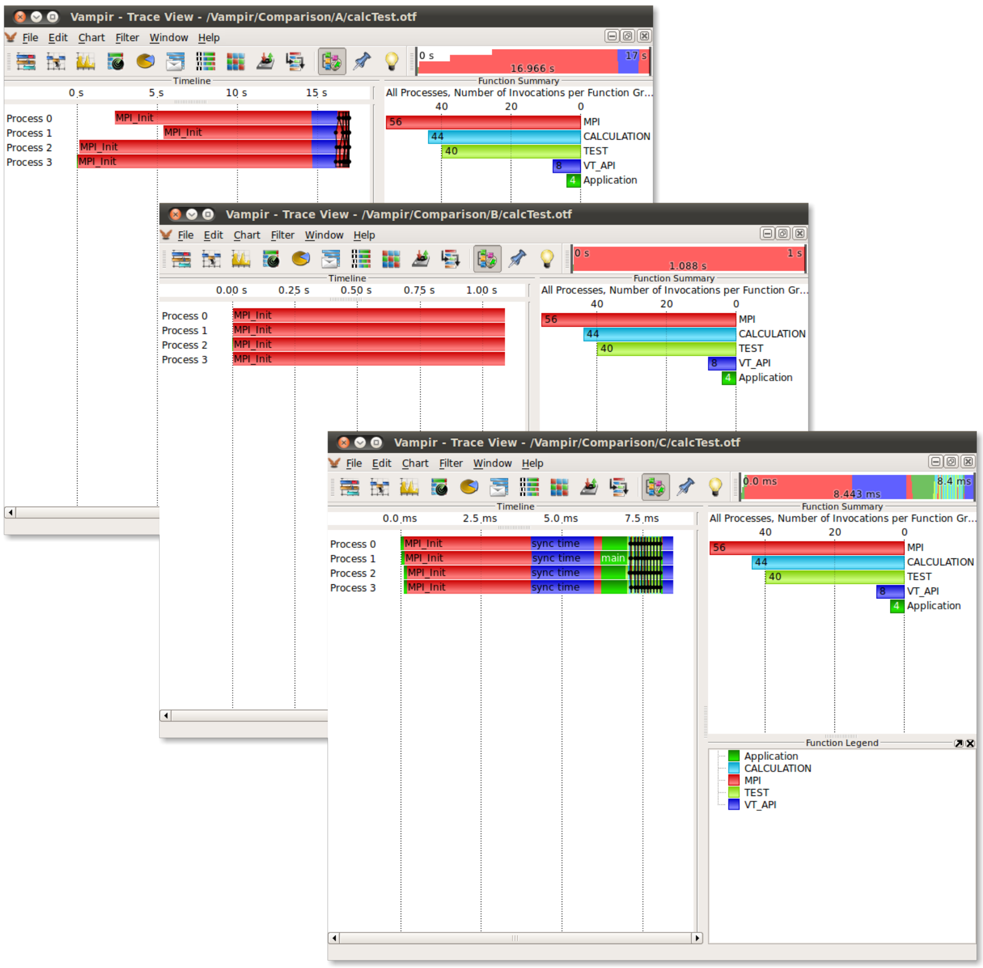
Figure 5.1: Compare View

All comparison features are provided in the “Compare View” window, depicted in Figure 5.1.

This section introduces the “Compare View” window and explains its usage. It illustrates the functionality with the help of screen shots. For this purpose the comparison of three trace files is demonstrated step by step. The three example trace files shown are measurements of one test application, performing ten iterations of simple calculations, run on three different machines.

5.1 Starting the Compare View

The fist step in order to compare trace files in *Vampir* is to load the respective files. Each trace file can be loaded one after another via “File → Open...” in the main menu. Figure 5.2 shows *Vampir* with three open trace files.



Vampir - Trace View - /Vampir/Comparison/A/calcTest.otf
File Edit Chart Filter Window Help
Timeline
Function Summary
All Processes, Number of Invocations per Function Gr...
Process 0
Process 1
Process 2
Process 3
MPI_Init
MPI
CALCULATION
TEST
VT_API
Application
Vampir - Trace View - /Vampir/Comparison/B/calcTest.otf
Vampir - Trace View - /Vampir/Comparison/C/calcTest.otf
sync time
main
Function Legend

Figure 5.2: Vampir with three open trace files

Now, the “Compare View” can be opened. This is done from the main menu by selecting “Window → Compare Traces”, see Figure 5.3.

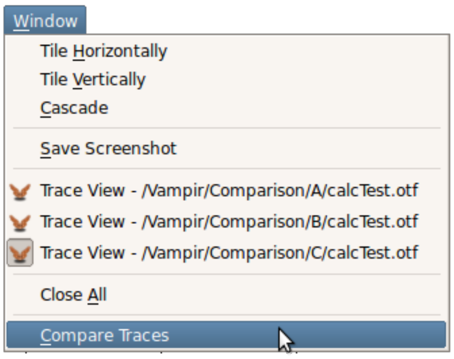

Window
Tile Horizontally
Tile Vertically
Cascade
Save Screenshot
Trace View - /Vampir/Comparison/A/calcTest.otf
Trace View - /Vampir/Comparison/B/calcTest.otf
Trace View - /Vampir/Comparison/C/calcTest.otf
Close All
Compare Traces

Figure 5.3: Vampir Window menu entries

Figure 5.4 shows the opened "Compare View". As can be seen by the navigation toolbars all three open trace files are included in the "Compare View". The files in the view are sharing one coupled zoom. The usage of charts and zooming in this view is described in Section 5.2.

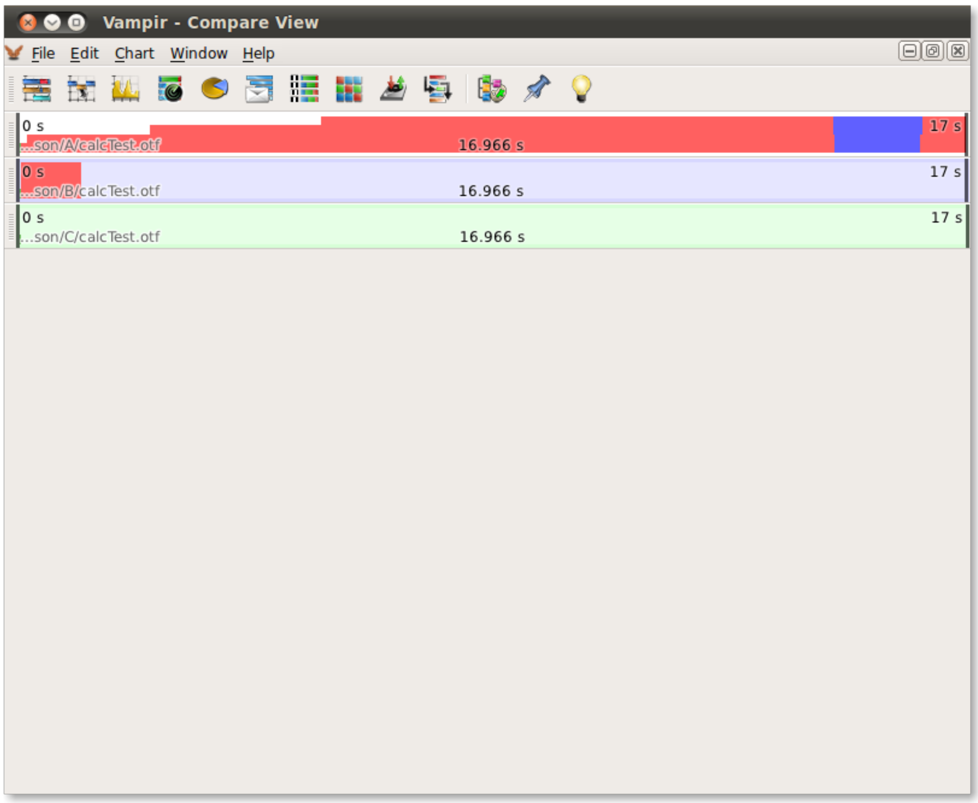


Figure 5.4: Open Compare View

5.2 Usage of Charts

For the comparison of performance metrics the “Compare View” provides all common charts of *Vampir*. In contrast to the ordinary “Trace View” the “Compare View” opens one chart instance for each trace file, i.e., with three open trace files, one click on the “Master Timeline” icon opens three ”Master Timeline” charts. Also, in order to distinguish the same charts between the trace files, all charts belonging to one trace file have a special individual background color. Figure 5.5 depicts a “Compare View” with open “Master Timeline”, “Process Timeline”, and “Function Summary” charts.

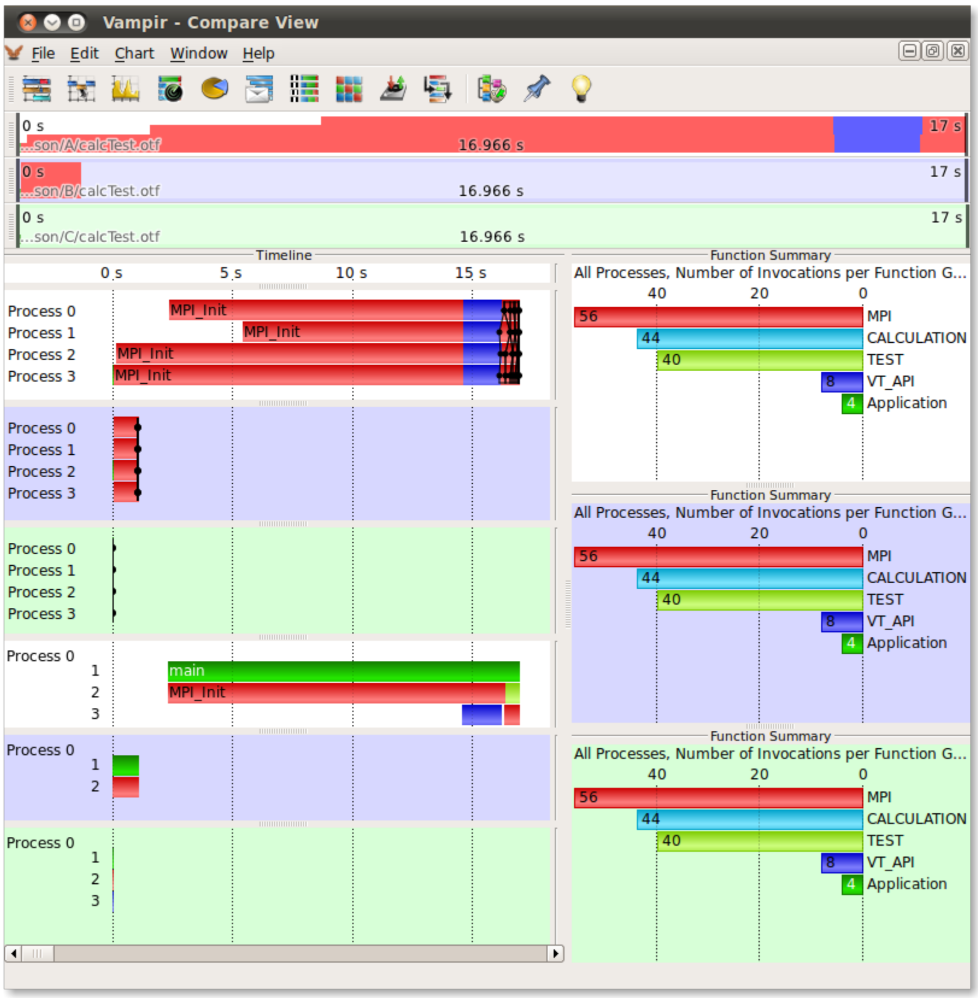


Figure 5.5: Compare View with open charts

All charts work the same way as in the “Trace View”. Due to the fact that the “Compare View” couples the zoom of all trace files, the charts can be used to compare performance characteristics.

As can be seen in Figure 5.5, trace A has the biggest duration time. The duration of

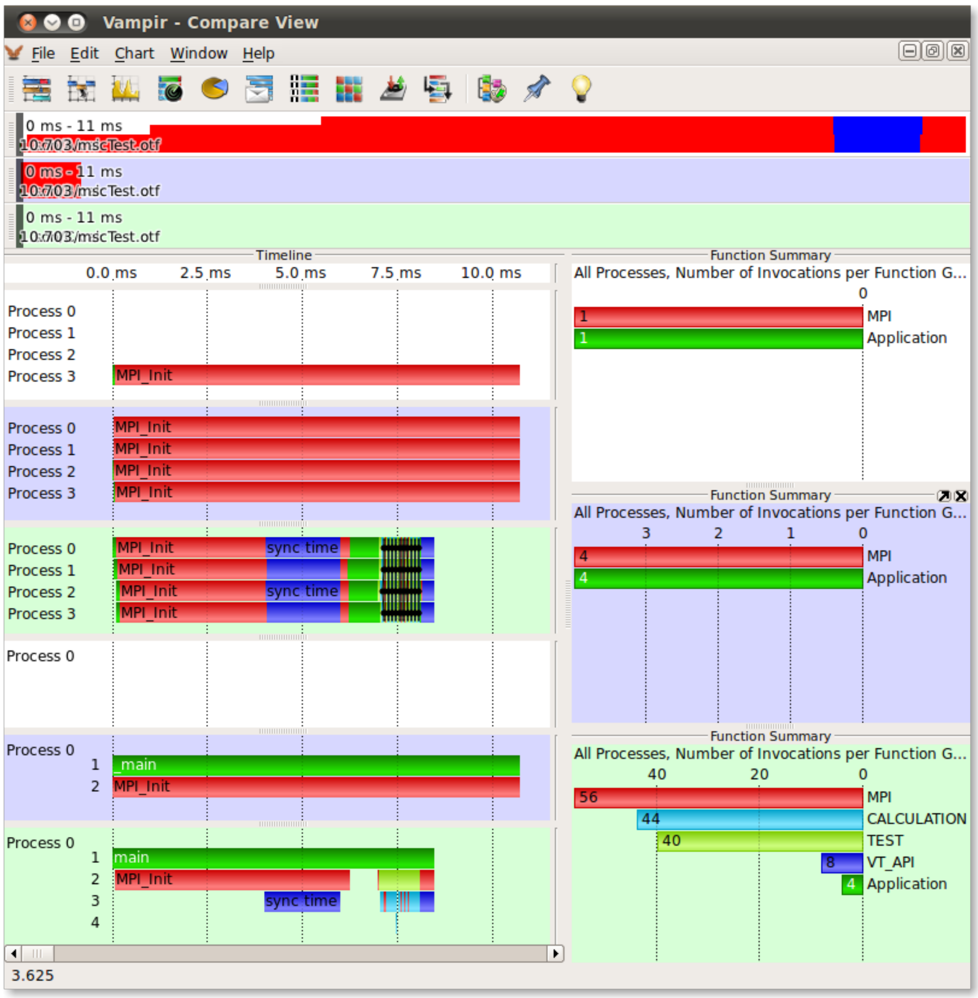


Figure 5.6: Zoom to compute iterations of trace C

trace C is so short that it is almost not visible. Zooming into the compute iterations of trace C would make them visible but would also only reveal the "MPI_Init" phase of trace A and B, see Figure 5.6. In order to compare the compute iterations the trace files need to be aligned properly. This process is described in Section 5.3.

5.3 Alignment of Multiple Trace Files

The "Compare View" functionality to shift individual trace files in time allows to compare areas of the data that did not occur at the same time. For instance, in order to compare the compute iterations of the three example trace files these areas need to be aligned to each other. This is required due to the fact that the initialization of the application took different times on the three machines.

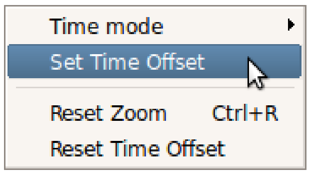


Figure 5.7: Context menu controlling the time offset

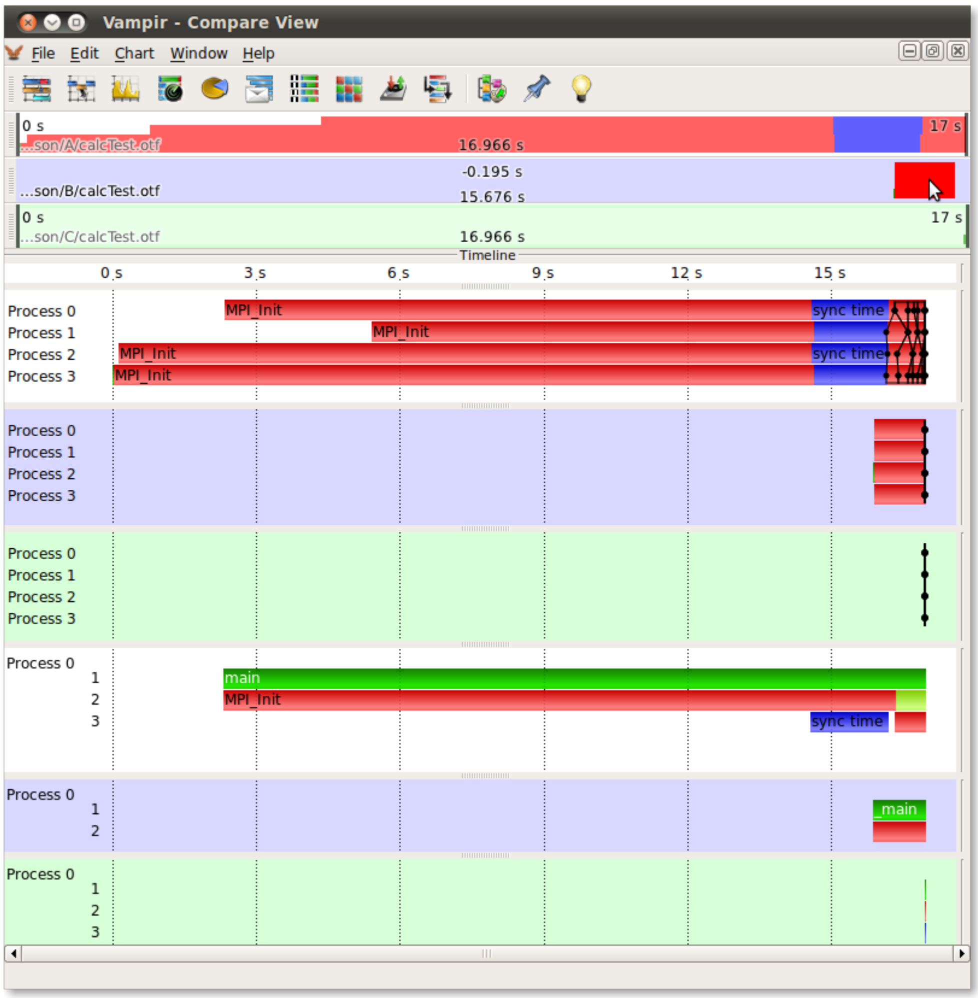


Figure 5.8: Alignment in the Navigation Toolbar

There are several ways to shift the trace files in time. One way is to use the context menu of the “Navigation Toolbar”. A right click on the toolbar reveals the menu shown in Figure 5.7. Here the entry “Set Time Offset” allows to manually set the time offset for the trace file. The entry “Reset Time Offset” resets the offset.

The easiest way to achieve a coarse alignment is to drag the trace file in the “Navigation Toolbar” itself. While holding the “Ctrl” (“Cmd” on Mac OS X) modifier key pressed the trace can be dragged to the desired position with the left mouse button. In Figure 5.8 the compute iterations of all example trace files are coarsely aligned.

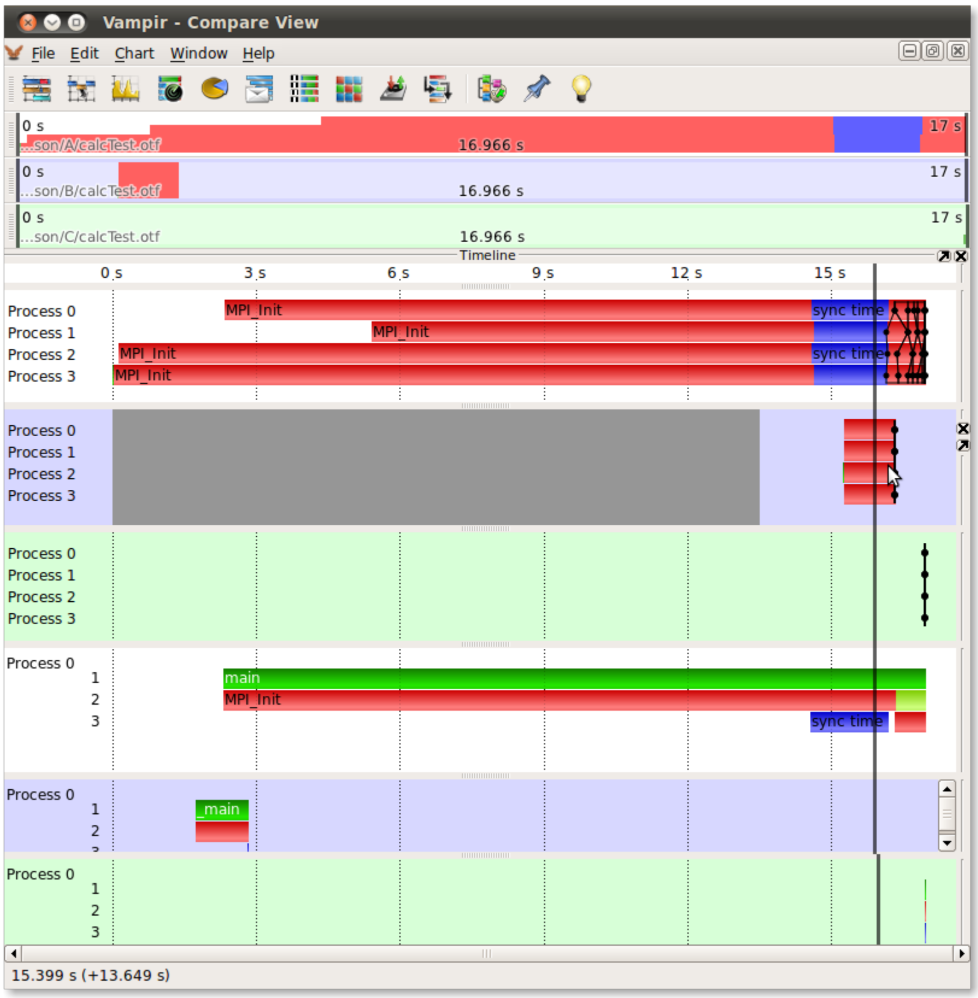
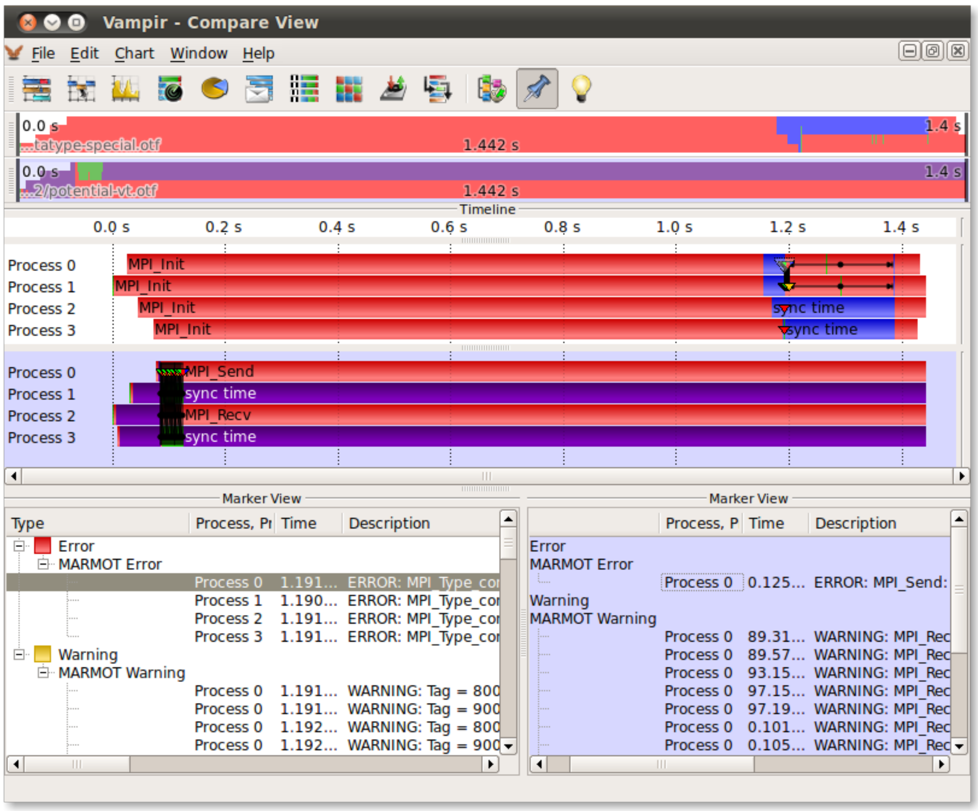


Figure 5.9: Alignment in the Master Timeline

After the coarse shifting a finer alignment can be done in the “Master Timeline”. Therefore the user needs to zoom into the area to compare. Then, while keeping the “Ctrl” (“Cmd” on Mac OS X) modifier key pressed, the trace can be dragged with the left mouse button in the “Master Timeline”. Figure 5.9 depicts the process of dragging trace B to the compute iterations of trace A. As can be seen in the Figure 5.9, although the initialization of trace A took the longest time, this machine was the fastest in computing the iterations.

5.4 Usage of Predefined Markers

The Open Trace Format (OTF) allows to define markers pointing to particular places of interest in the trace data. These markers can be used to navigate in the trace files. For trace file comparison markers are interesting due to their potential to quickly locate places in large trace data. With the help of markers it is possible to find the same location in multiple trace files with just a few clicks.



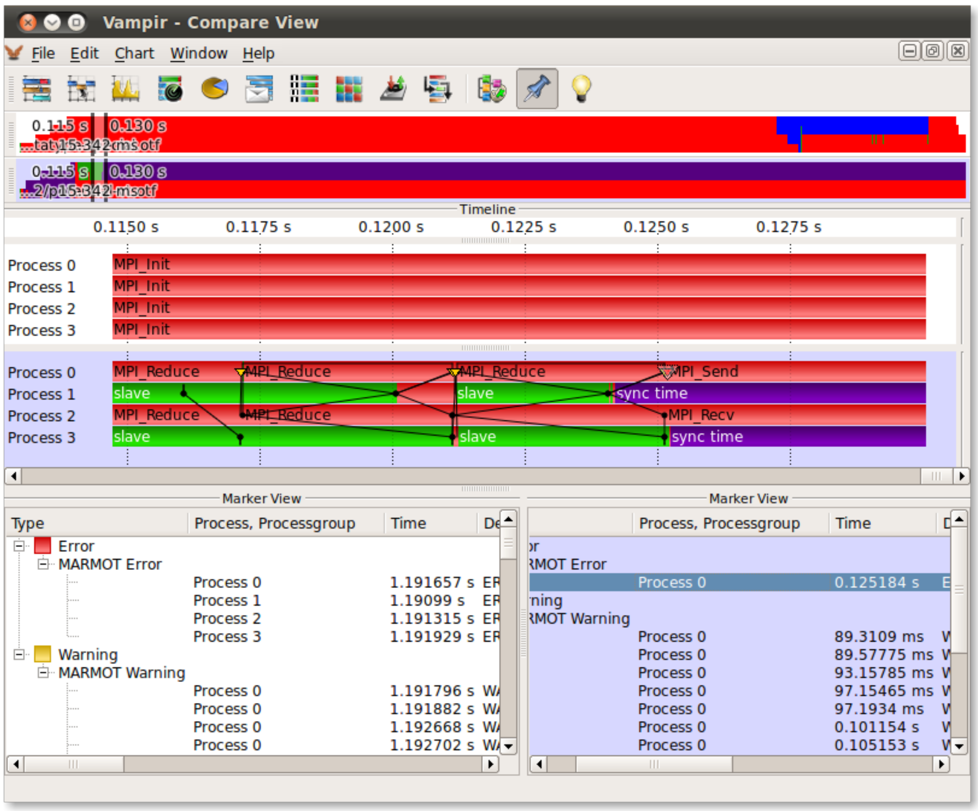
Type	Process, Pr	Time	Description
🟥 Error			
MARMOT Error			
	Process 0	1.191...	ERROR: MPI_Type_cor
	Process 1	1.190...	ERROR: MPI_Type_cor
	Process 2	1.191...	ERROR: MPI_Type_cor
	Process 3	1.191...	ERROR: MPI_Type_cor
🟨 Warning			
MARMOT Warning			
	Process 0	1.191...	WARNING: Tag = 800
	Process 0	1.191...	WARNING: Tag = 900
	Process 0	1.192...	WARNING: Tag = 800
	Process 0	1.192...	WARNING: Tag = 900

	Process, P	Time	Description
Error			
MARMOT Error			
	Process 0	0.125...	ERROR: MPI_Send:
Warning			
MARMOT Warning			
	Process 0	89.31...	WARNING: MPI_Rec
	Process 0	89.57...	WARNING: MPI_Rec
	Process 0	93.15...	WARNING: MPI_Rec
	Process 0	97.15...	WARNING: MPI_Rec
	Process 0	97.19...	WARNING: MPI_Rec
	Process 0	0.101...	WARNING: MPI_Rec
	Process 0	0.105...	WARNING: MPI_Rec

Figure 5.10: Open Marker View

First step in order to use markers is to open the “Marker View”. Figure 5.10 shows a “Compare View” with an open “Marker View” for each trace file. After a click on one marker in the “Marker View” the selected marker is highlighted in the “Master Timeline” and the “Process Timeline”.

Another way to navigate to a marker in the timeline displays is to use the *Vampir* zoom. If the user zoomed in the “Master Timeline” or “Process Timeline” into the desired zooming level, then a click on a marker in the “Marker View” will shift the timeline zoom to the marker position. Thus, the marker appears in the center of the timeline display, see Figure 5.11.



Type	Process, Processgroup	Time	De
Error			
MARMOT Error			
	Process 0	1.191657 s	ER
	Process 1	1.19099 s	ER
	Process 2	1.191315 s	ER
	Process 3	1.191929 s	ER
Warning			
MARMOT Warning			
	Process 0	1.191796 s	WA
	Process 0	1.191882 s	WA
	Process 0	1.192668 s	WA
	Process 0	1.192702 s	WA

	Process, Processgroup	Time	D
or			
RMOT Error			
	Process 0	0.125184 s	E
ning			
RMOT Warning			
	Process 0	89.3109 ms	W
	Process 0	89.57775 ms	W
	Process 0	93.15785 ms	W
	Process 0	97.15465 ms	W
	Process 0	97.1934 ms	W
	Process 0	0.101154 s	W
	Process 0	0.105153 s	W

Figure 5.11: Jump to marker in the Master Timeline

6 Customization

The appearance of the trace file and various other application settings can be altered in the preferences accessible via the main menu entry “File → Preferences”. Settings concerning the trace file itself, e.g. layout or function group colors are saved individually next to the trace file in a file with the ending “.vsettings”. This way it is possible to adjust the colors for individual trace files without interfering with others.

The options “Import Preferences” and “Export Preferences” provide the loading and saving of preferences of arbitrary trace files.

6.1 General Preferences

The “General” settings allow to change application and trace specific values.

“Show time as” decides whether the time format for the trace analysis is based on seconds or ticks.

With the “Automatically open context view” option disabled *Vampir* does not open the context view after the selection of an item, like a message or function.

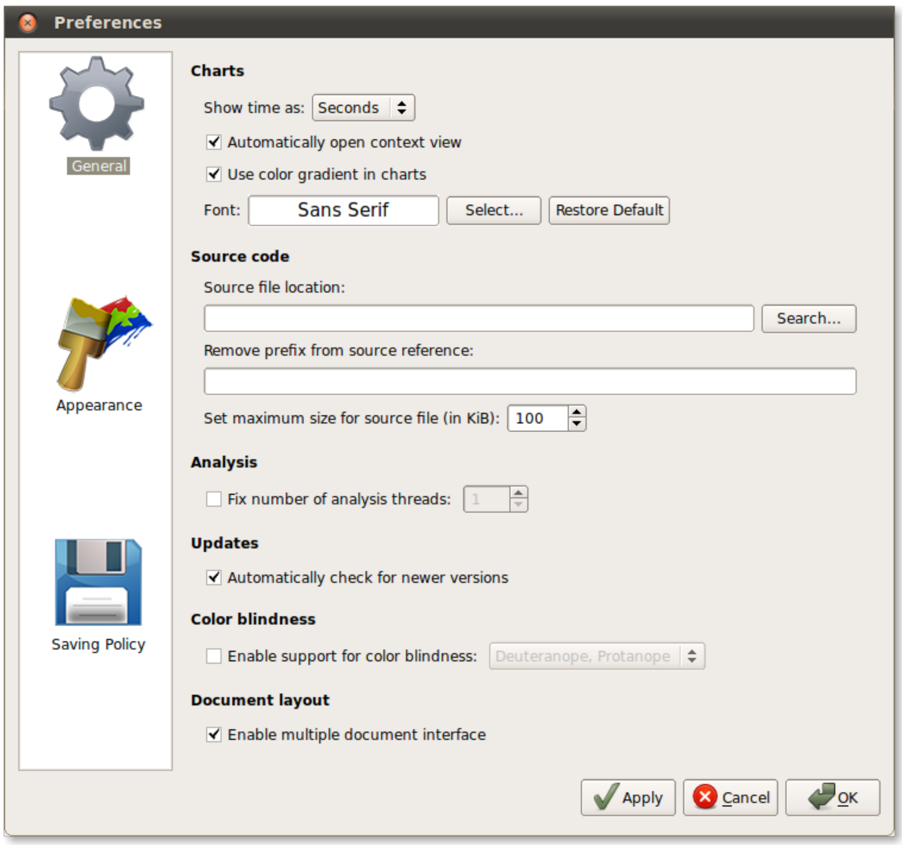
“Use color gradient in charts” allows to switch off the color gradient used in the performance charts.

The next option allows to change the style and size of the font.

“Show source code” enables the internal source code viewer. This viewer shows the source code corresponding to selected locations in the trace file. In order to open a source file first click on the intended function in the “Master Timeline” and then on the source code path in the “Context View”. For the source code location to work properly, you need a trace file with source code location support. The path to the source file can be adjusted in the “Preferences” dialog. A limit for the size of the source file to be opened can be set, too.

In the “Analysis” section the number of analysis threads can be chosen. If this option is disabled, *Vampir* determines the number automatically by the number of cores, e.g. two analysis threads on a dual-core machine.

In the “Updates” section the user can decide if *Vampir* should check automatically for new versions.



Charts

Show time as: Seconds

[✓] Automatically open context view

[✓] Use color gradient in charts

Font: Sans Serif

Source code

Source file location:

Remove prefix from source reference:

Set maximum size for source file (in KiB): 100

Analysis

[ ] Fix number of analysis threads: 1

Updates

[✓] Automatically check for newer versions

Color blindness

[ ] Enable support for color blindness: Deuteranope, Protanope

Document layout

[✓] Enable multiple document interface

Figure 6.1: General Settings

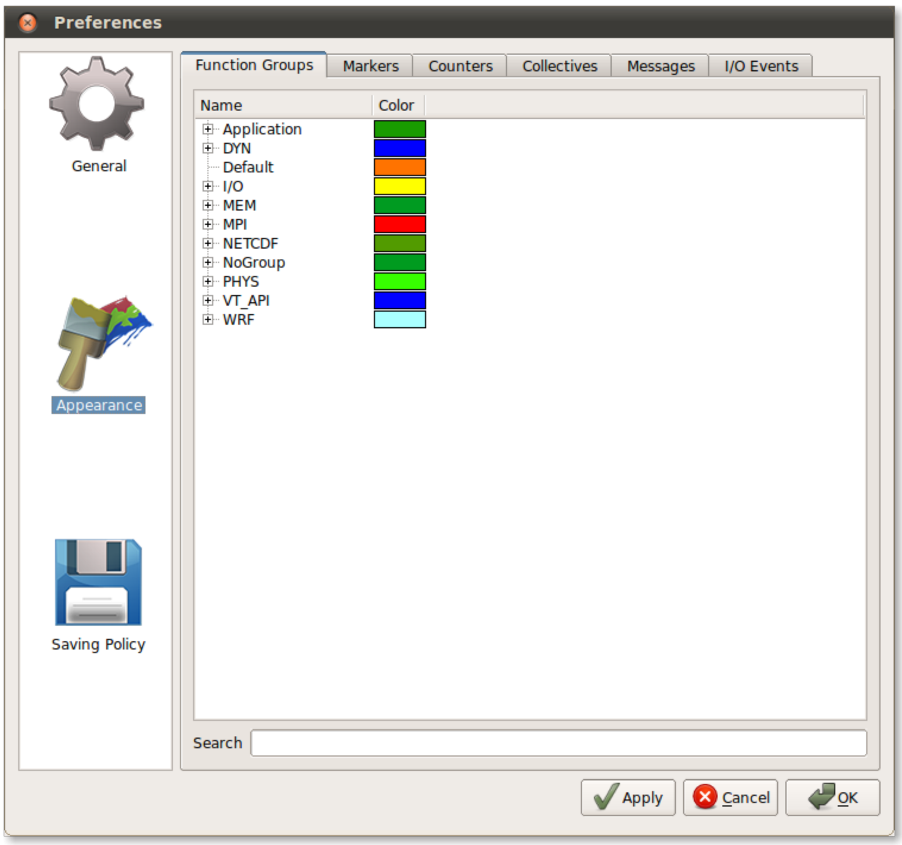
Vampir also features a color blindness support mode.

On Linux systems there is also the “Document layout” option available. If this option is enabled all open “Trace View” windows need to stay in one main window. If it is disabled, the “Trace View” windows can be moved freely over the Desktop.

6.2 Appearance

In the “Appearance” settings of the “Preferences” dialog there are six different objects for which the color options can be changed. The functions/function groups, markers, counters, collectives, messages, and I/O events. Choose an entry and click on its color to make a modification. A color picker dialog opens where it is possible to adjust the color. For messages and collectives a change of the line width is also available.

In order to quickly find the desired item a search box is provided at the bottom of the dialog.



Name	Color
Application	🟩
DYN	🟦
Default	🟧
I/O	🟨
MEM	🟩
MPI	🟥
NETCDF	🟩
NoGroup	🟩
PHYS	🟩
VT_API	🟦
WRF	🟦

Figure 6.2: Appearance Settings

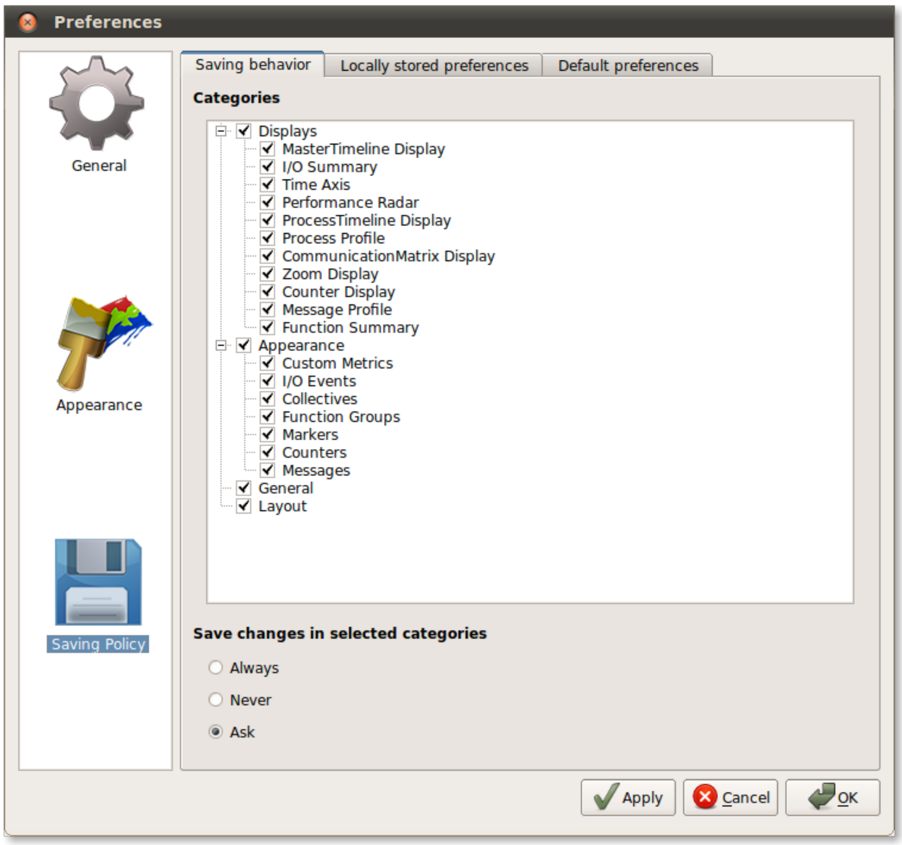
6.3 Saving Policy

Vampir detects whenever changes to the various settings are made. In the “Saving Policy” dialog it is possible to adjust the saving behavior of the different components to the own needs.

In the dialog “Saving Behavior” you tell *Vampir* what to do in the case of changed preferences. The user can choose the categories of settings, e.g., the layout, that should be affected by the selected behavior. Possible options are that the application automatically “Always” or “Never” saves changes. The default option is to have *Vampir* asking you whether to save or discard changes.

Usually the settings are stored in the folder of the trace file. If the user has no write access to it, it is possible to place them alternatively in the “Application Data Folder”. All such stored settings are listed in the tab “Locally Stored Preferences” with creation and modification date.

Note: On loading *Vampir* always favors settings in the “Application Data Folder”.



Categories

- ✓ Displays
  - ✓ MasterTimeline Display
  - ✓ I/O Summary
  - ✓ Time Axis
  - ✓ Performance Radar
  - ✓ ProcessTimeline Display
  - ✓ Process Profile
  - ✓ CommunicationMatrix Display
  - ✓ Zoom Display
  - ✓ Counter Display
  - ✓ Message Profile
  - ✓ Function Summary
- ✓ Appearance
  - ✓ Custom Metrics
  - ✓ I/O Events
  - ✓ Collectives
  - ✓ Function Groups
  - ✓ Markers
  - ✓ Counters
  - ✓ Messages
- ✓ General
- ✓ Layout

Save changes in selected categories

Always

Never

[✓ Ask]

Figure 6.3: Saving Policy Settings

“Default Preferences” offers to save preferences of the current trace file as default settings. Then they are used for trace files without settings. Another option is to restore the default settings. Then the current preferences of the trace file are reverted.

7 A Use Case

This chapter explains by example how *Vampir* can be used to discover performance problems in your code and how to correct them.

7.1 Introduction

In many cases the *Vampir* suite has been successfully applied to identify performance bottlenecks and assist their correction. To show in which ways the provided toolset can be used to find performance problems in program code, one optimization process is illustrated in this chapter. The following example is a three-part optimization of a weather forecast model including simulation of cloud microphysics. Every run of the code has been performed on 100 cores with manual function instrumentation, MPI communication instrumentation, and recording of the number of L2 cache misses.

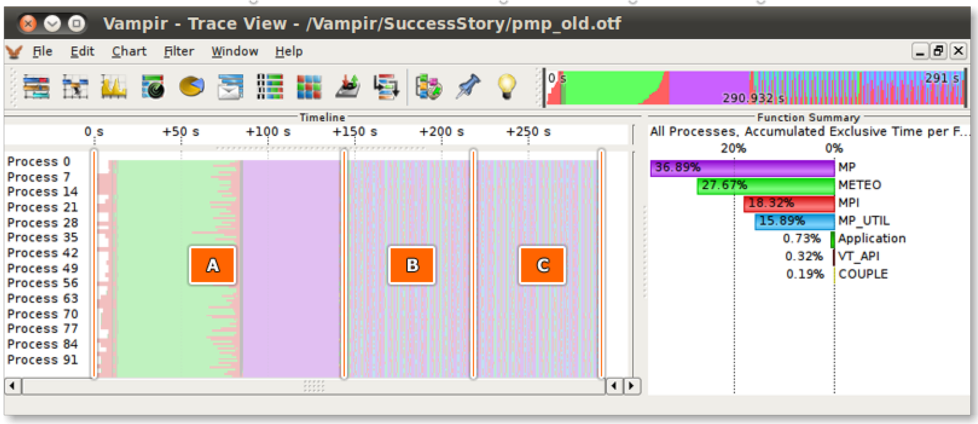


Figure 7.1: Master Timeline and Function Summary showing an overview of the program run

Getting a grasp of the program's overall behavior is a reasonable first step. In Figure 7.1 *Vampir* has been set up to provide such a high-level overview of the model's code. This layout can be achieved through two simple manipulations. Set up the Master Timeline to adjust the process bar height to fit the chart height. All 100 processes are now arranged into one view. Likewise, change the event category in the Function

Summary to show function groups. This way the many functions are condensed into fewer function groups.

One run of the instrumented program took 290 seconds to finish. The first half of the trace (Figure 7.1 A) is the initialization part. Processes get started and synced, input is read and distributed among these processes. The preparation of the cloud microphysics (function group: MP) is done here as well.

The second half is the iteration part, where the actual weather forecasting takes place. In a normal weather simulation this part would be much larger. But in order to keep the recorded trace data and the overhead introduced by tracing as small as possible only a few iterations have been recorded. This is sufficient since they are all doing the same work anyway. Therefore the simulation has been configured to only forecast the weather 20 seconds into the future. The iteration part consists of two ”large” iterations (Figure 7.1 B and C), each calculating 10 seconds of forecast. Each of these in turn is partitioned into several ”smaller” iterations.

For our observations we focus on only two of these small, inner iterations, since this is the part of the program where most of the time is spent. The initialization work does not increase with a higher forecast duration and would only take a relatively small amount of time in a real world run. The constant part at the beginning of each large iteration takes less than a tenth of the whole iteration time. Therefore, by far the most time is spent in the small iterations. Thus they are the most promising candidates for optimization.

All screenshots starting with Figure 7.2 are in a before-and-after fashion to point out what changed by applying the specific improvements.

7.2 Identified Problems and Solutions

7.2.1 Computational Imbalance

A varying size of work packages (thus varying processing time of this work) means waiting time in subsequent synchronization routines. This section points out two easy ways to recognize this problem.

Problem

As can be seen in Figure 7.2 each occurrence of the MICROPHYSICS-routine (purple color) starts at the same time on all processes inside one iteration, but takes between 1.7 and 1.3 seconds to finish. This imbalance leads to idle time in subsequent synchronization calls on the processes 1 to 4, because they have to wait for process 0 to finish its work (marked parts in Figure 7.2). This is wasted time which could be used for

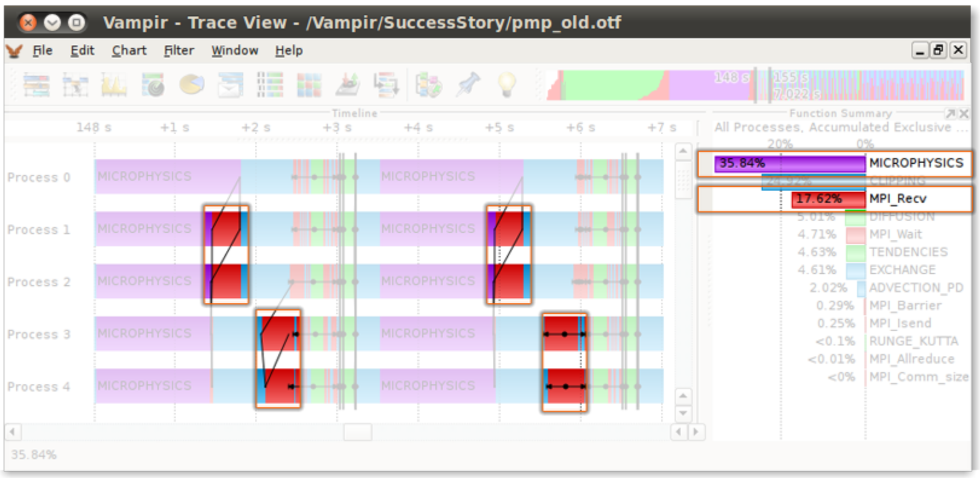


Figure 7.2: Before Tuning: Master Timeline and Function Summary identifying MICROPHYSICS (purple color) as predominant and unbalanced

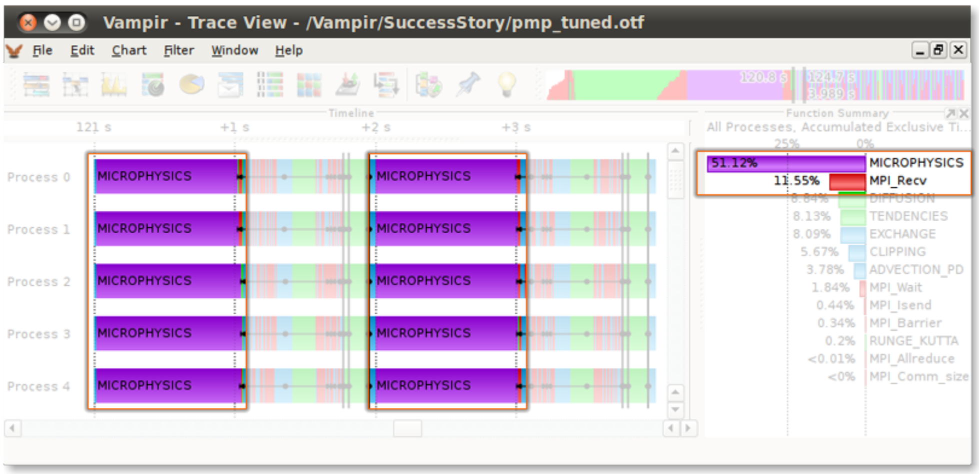


Figure 7.3: After Tuning: Timeline and Function Summary showing an improvement in communication behavior

computational work, if all MICROPHYSICS-calls would have the same duration. Another hint at this overhead in synchronization is the fact that the MPI receive routine uses 17.6% of the time of one iteration (Function Summary in Figure 7.2).

Solution

To even out this asymmetry the code which determines the size of the work packages for each process had to be changed. To achieve the desired effect an improved version of the domain decomposition has been implemented. Figure 7.3 shows that all occurrences of the MICROPHYSICS-routine are vertically aligned, thus balanced. Additionally the MPI receive routine calls are now clearly smaller than before. Comparing the Function Summary of Figure 7.2 and Figure 7.3 shows that the relative time spent in MPI receive has been decreased, and in turn the time spent inside MICROPHYSICS has been increased greatly. This means that we now spend more time computing and less time communicating, which is exactly what we want.

7.2.2 Serial Optimization

Inlining of frequently called functions and elimination of invariant calculations inside loops are two ways to improve the serial performance. This section shows how to detect candidate functions for serial optimization and suggests measures to speed them up.

Problem

All performance charts in *Vampir* show information of the time span currently selected in the timeline. Thus the most time-intensive routine of one iteration can be determined by zooming into one or more iterations and having a look at the Function Summary. The function with the largest bar takes up the most time. In this example (Figure 7.2) the MICROPHYSICS-routine can be identified as the most costly part of an iteration. Therefore it is a good candidate for gaining speedup through serial optimization techniques.

Solution

In order to get a fine-grained view of the MICROPHYSICS-routine's inner workings we had to trace the program using full function instrumentation. Only then it was possible to inspect and measure subroutines and subsubroutines of MICROPHYSICS. This way the most time consuming subroutines have been spotted, and could be analyzed for optimization potential.

The review showed that there were a couple of small functions which were called a lot. So we simply inlined them. With *Vampir* you can determine how often a functions is called by changing the metric of the Function Summary to the number of invocations.

The second inefficiency we discovered had been invariant calculations being done inside loops. So we just moved them in front of the respective loops.

Figure 7.3 sums up the tuning of the computational imbalance and the serial optimization. In the timeline you can see that the duration of the MICROPHYSICS-routine is now equal among all processes. Through serial optimization the duration has been decreased from about 1.5 to 1.0 second. A decrease in duration of about 33% is quite good given the simplicity of the changes done.

7.2.3 High Cache Miss Rate

The latency gap between cache and main memory is about a factor of 8. Therefore optimizing for cache usage is crucial for performance. If you don't access your data in a linear fashion as the cache expects, so called cache misses occur and the specific instructions have to suspend execution until the requested data arrives from main memory. A high cache miss rate therefore indicates that performance might be improved through reordering of the memory access pattern to match the cache layout of the platform.

Problem

As can be seen in the Counter Data Timeline (Figure 7.4) the CLIPPING-routine (light blue) causes a high amount of L2 cache misses. Also its duration is long enough to make it a candidate for inspection. What caused these inefficiencies in cache usage were nested loops, which accessed data in a very random, non-linear fashion. Data access can only profit from cache if subsequent read calls access data in the vicinity of the previously accessed data.

Solution

After reordering the nested loops to match the memory order, the tuned version of the CLIPPING-routine now needs only a fraction of the original time. (Figure 7.5)

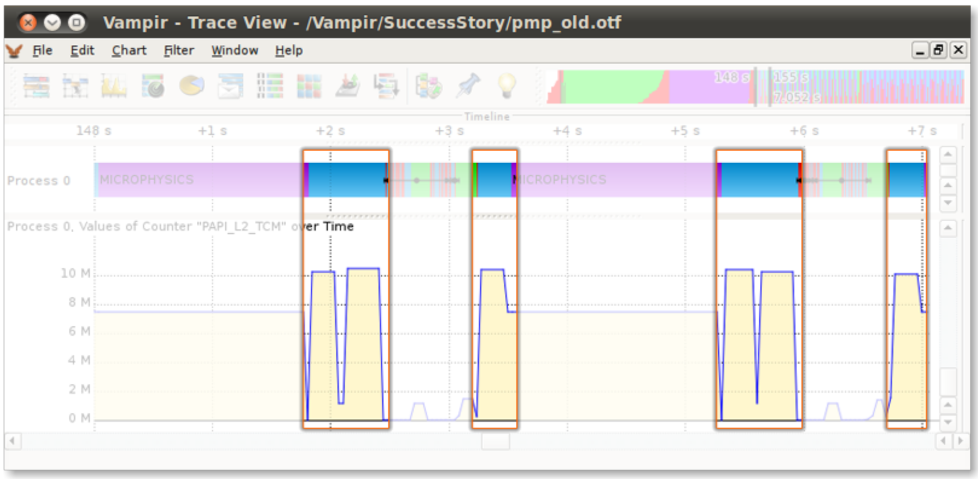

Vampir - Trace View - /Vampir/SuccessStory/pmp_old.otf
File Edit Chart Filter Window Help
Timeline
148 s
+1 s
+2 s
+3 s
+4 s
+5 s
+6 s
+7 s
Process 0
MICROPHYSICS
Process 0, Values of Counter "PAPI_L2_TCM" over Time
10 M
8 M
6 M
4 M
2 M
0 M

Figure 7.4: Before Tuning: Counter Data Timeline revealing a high amount of L2 cache misses inside the CLIPPING-routine (light blue)

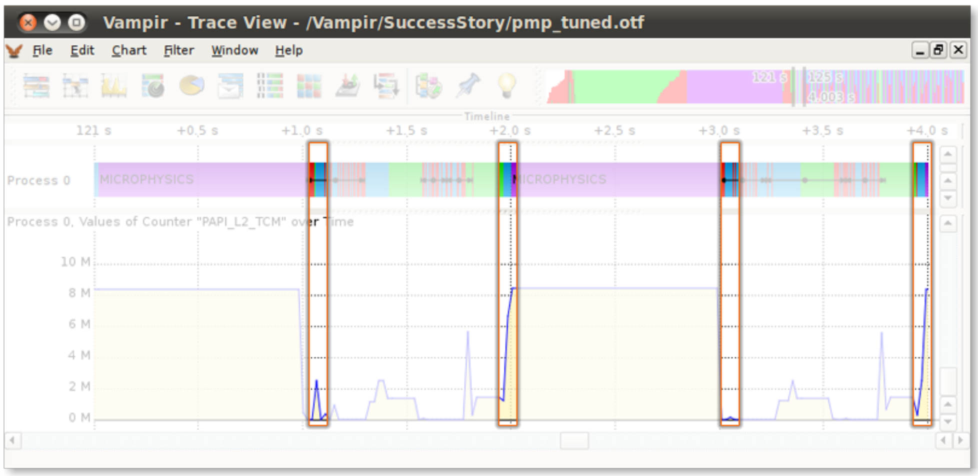

Vampir - Trace View - /Vampir/SuccessStory/pmp_tuned.otf
File Edit Chart Filter Window Help
Timeline
121 s
+0.5 s
+1.0 s
+1.5 s
+2.0 s
+2.5 s
+3.0 s
+3.5 s
+4.0 s
Process 0
MICROPHYSICS
Process 0, Values of Counter "PAPI_L2_TCM" over Time
10 M
8 M
6 M
4 M
2 M
0 M

Figure 7.5: After Tuning: Visible improvement of the cache usage

7.3 Conclusion

By using the *Vampir* toolkit, three problems have been identified. As a consequence of addressing each problem, the duration of one iteration has been decreased from 3.5 seconds to 2.0 seconds.

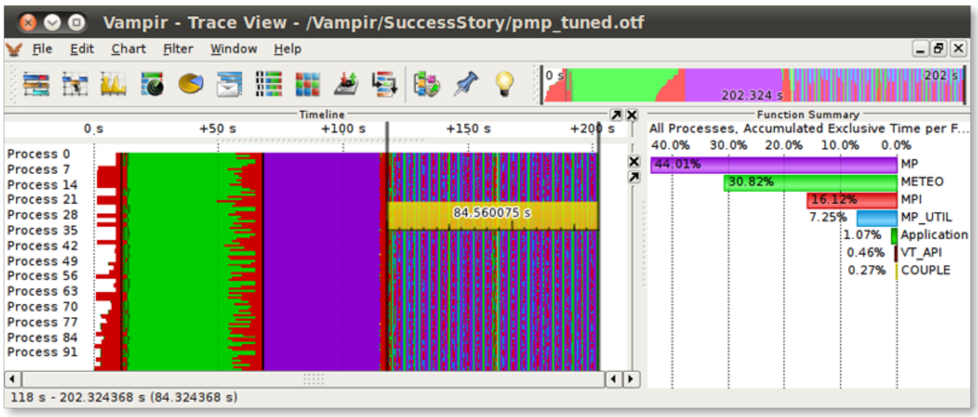


Figure 7.6: Overview showing a significant overall improvement

As is shown by the *Ruler* (Chapter 4.1) in Figure 7.6 two large iterations now take 84 seconds to finish. Whereas at first (Figure 7.1) it took roughly 140 seconds, making a total speed gain of 40%.

This huge improvement has been achieved by using the insight into the program's runtime behavior, provided by the *Vampir* toolkit, to optimize the inefficient parts of the code.